



Data Analytics Server

Documentation



WSO2 Data Analytics Server

Documentation

Version 3.1.0

Table of Contents

1. WSO2 Data Analytics Server Documentation	7
1.1 About DAS	7
1.1.1 Introducing DAS	8
1.1.2 DAS Features	10
1.1.3 Architecture	13
1.1.4 About this Release	15
1.2 Getting Started	15
1.2.1 Quick Start Guide	16
1.2.2 Downloading the Product	66
1.2.3 Installation Prerequisites	66
1.2.4 Installing the Product	69
1.2.4.1 Installing on Linux	69
1.2.4.2 Installing on Windows	71
1.2.4.3 Installing as a Windows Service	76
1.2.4.4 Installing as a Linux Service	81
1.2.5 Building from Source	83
1.2.6 Upgrading from WSO2 DAS 3.0.1	84
1.2.7 Running the Product	97
1.2.8 WSO2 DAS Production Setup Checklist	101
1.3 User Guide	102
1.3.1 Understanding Event Streams and Event Tables	103
1.3.1.1 Configuring Data Persistence	112
1.3.1.1.1 Implementation With Different Database Types	117
1.3.1.1.2 Accessing Persisted Data	130
1.3.1.1.3 Switching Databases without Migrating Data	132
1.3.2 Collecting Data	132
1.3.2.1 Publishing Data to DAS	132
1.3.2.1.1 Publishing Data Using Event Simulation	132
1.3.2.1.2 Publishing Data Using Java Client Through Thrift or Binary	136
1.3.2.1.3 How to Publish Data Through Other Protocols	145
1.3.2.2 Persisting Data for Batch Analytics	152
1.3.2.3 Configuring DAS to Receive Data	153
1.3.2.3.1 Configuring Event Receivers	153
1.3.2.3.2 Input Mapping Types	195
1.3.2.3.3 Building Custom Event Receivers	205
1.3.2.4 Configuring Received Data	212
1.3.3 Analyzing Data	213
1.3.3.1 Realtime Analytics Using Siddhi	213
1.3.3.1.1 Creating a Standalone Execution Plan	213
1.3.3.1.2 Creating a STORM Based Distributed Execution Plan	218
1.3.3.1.3 Siddhi Query Language	219
1.3.3.2 Interactive Analytics	357
1.3.3.2.1 Persisting Data for Interactive Analytics	357
1.3.3.2.2 Configuring Indexes	358
1.3.3.2.3 Data Explorer	381

1.3.3.2.4 Activity Explorer	384
1.3.3.2.5 Query Language Reference	389
1.3.3.3 Batch Analytics Using Spark SQL	391
1.3.3.3.1 Batch Analytics Console	392
1.3.3.3.2 Scheduling Batch Analytics Scripts	392
1.3.3.3.3 Spark Query Language	396
1.3.3.3.4 Publishing Events Using Apache Spark	406
1.3.3.3.5 Creating Spark User Defined Functions	407
1.3.3.3.6 Creating Spark User Defined Aggregate Functions	409
1.3.3.3.7 Spark Troubleshooting	413
1.3.3.3.8 Incremental Processing	413
1.3.3.3.9 Monitoring Spark Performance	417
1.3.3.4 Predictive Analytics	417
1.3.3.4.1 Using a ML Model Within WSO2 CEP	418
1.3.3.4.2 Using a ML Model Within WSO2 ESB	418
1.3.4 Communicating Results	418
1.3.4.1 Visualizing Results	418
1.3.4.1.1 Analytics Dashboard	418
1.3.4.1.2 Geo Dashboard	424
1.3.4.2 Creating Alerts	426
1.3.4.2.1 Event Publisher Types	436
1.3.4.2.2 Output Mapping Types	496
1.3.4.3 Communicating Results Through REST API	508
1.3.4.4 Analytics JavaScript (JS) API	508
1.3.4.4.1 Retrieving the List of Tables of All Record Stores via JS API	509
1.3.4.4.2 Retrieving All Record Stores via JS API	510
1.3.4.4.3 Retrieving the Record Store of a Given Table via JS API	510
1.3.4.4.4 Checking if a Given Table Exists via JS API	511
1.3.4.4.5 Clearing Indexed Data of a Given Table via JS API	512
1.3.4.4.6 Retrieving Records Based on a Time Range via JS API	512
1.3.4.4.7 Retrieving Records Matching the Given Primary Key Combination via JS API	513
1.3.4.4.8 Retrieving Records of Given Record IDs via JS API	514
1.3.4.4.9 Retrieving the Total Record Count of a Table via JS API	515
1.3.4.4.10 Retrieving the Number of Records Matching the Given Search Query via JS API	516
1.3.4.4.11 Retrieving All Records Matching the Given Search Query via JS API	516
1.3.4.4.12 Retrieving the Schema of a Table via JS API	518
1.3.4.4.13 Checking if the Given Records Store Supports Pagination via JS API	519
1.3.4.4.14 Tracking the Indexing Process Completion via JS API	519
1.3.4.4.15 Drilling Down Through Categories via JS API	520
1.3.4.4.16 Retrieving Specific Records through a Drill Down Search via JS API	521
1.3.4.4.17 Retrieving the Number of Records Matching the Drill Down Criteria via JS API	523
1.3.4.4.18 Adding an Event Stream Definition via JS API	524
1.3.4.4.19 Publishing Events to WSO2 DAS via JS API	525
1.3.4.4.20 Retrieving an Existing Event Definition via JS API	526
1.3.4.4.21 Tracking the Indexing Process Completion of a Table via JS API	527
1.3.4.4.22 Retrieving Aggregated Values of Given Records via JS API	528
1.3.5 Packaging Artifacts as a C-App Archive	529
1.3.6 Debugging	541

1.3.6.1 Event Statistics	541
1.3.6.2 Event Tracer	544
1.3.6.3 Siddhi Try It Tool	545
1.3.7 Managing DAS Artifacts via Template Manager	547
1.3.7.1 Configuring a Template for Template Manager	548
1.3.7.2 Using Templates	564
1.4 Admin Guide	569
1.4.1 Spark Configurations	570
1.4.2 Enabling/Disabling Selected DAS Components	576
1.4.3 Purging Data	578
1.4.4 Analytics Migration Tool	581
1.4.5 Analytics Data Backup / Restore Tool	584
1.4.6 Performance Tuning	587
1.4.7 Configuration Guide	597
1.4.7.1 Registry	598
1.4.7.1.1 Introduction to Registry	598
1.4.7.1.2 Managing the Registry	598
1.4.7.1.3 Searching the Registry	613
1.4.7.1.4 Sharing Registry Space Among Multiple Products	615
1.4.7.2 User Management	634
1.4.7.2.1 Introduction to User Management	635
1.4.7.2.2 Adding and Managing Users and Roles	636
1.4.7.2.3 Realm Configuration	642
1.4.7.2.4 Changing the RDBMS	646
1.4.7.2.5 Configuring Primary User Stores	647
1.4.7.2.6 Configuring Secondary User Stores	663
1.4.7.3 Feature Management	666
1.4.7.3.1 Introduction to Feature Management	666
1.4.7.3.2 Installing and Managing Features	666
1.4.7.3.3 Recovering from Unsuccessful Feature Installation	670
1.4.7.4 Logging	671
1.4.7.5 Datasources	674
1.4.7.5.1 Configuring an RDBMS Datasource	677
1.4.7.5.2 Configuring a Cassandra Datasource	730
1.4.7.5.3 Configuring a HBase Datasource	730
1.4.7.5.4 Configuring a HDFS Datasource	731
1.4.7.5.5 Configuring a Custom Datasource	732
1.4.7.6 Server Roles for C-Apps	734
1.4.7.6.1 Introduction to Server Roles	734
1.4.7.6.2 Adding a Server Role	735
1.4.7.6.3 Deleting a Server Role	736
1.4.7.6.4 Transports	737
1.4.7.7 Scheduling Tasks	753
1.4.7.8 Security	755
1.4.7.8.1 Fixing Security Vulnerabilities	755
1.4.7.8.2 Enabling Java Security Manager	756
1.4.7.8.3 Setting up Keystores	759
1.4.8 Connecting a DAS Instance to an Existing External Apache Spark Cluster	769

1.4.9 Storing Index Data	773
1.4.10 Configuring Single Sign-On for WSO2 DAS	775
1.4.11 Deployment and Clustering	782
1.4.11.1 Spark Deployment Patterns	785
1.4.12 Working with Product Specific Analytics Profiles	787
1.4.13 Supporting Different Transports	795
1.4.14 Installing WSO2 GPL Features	797
1.4.15 Changing the Host Name	798
1.4.16 Product Administration	799
1.5 Samples	802
1.5.1 Real Time Samples	803
1.5.1.1 Setting Up Real Time Samples	803
1.5.1.2 WSO2 DAS Real Time Samples	810
1.5.1.2.1 Samples on Receiving Events	816
1.5.1.2.2 Samples on Processing Events	858
1.5.1.2.3 Samples on Publishing Events	909
1.5.2 Sending Notifications Through Published Events Using Spark	950
1.5.3 Analyzing HTTPD Logs	958
1.5.4 Analyzing Smart Home Data	962
1.5.5 Analyzing Realtime Service Statistics	966
1.5.6 Analyzing Wikipedia Data	968
1.6 Reference Guide	975
1.6.1 REST APIs for Analytics Data Service	976
1.6.1.1 Analytics REST API Guide	976
1.6.1.1.1 Checking if a Given Table Exists via REST API	977
1.6.1.1.2 Retrieving the List of Tables of All Record Stores via REST API	978
1.6.1.1.3 Retrieving Records Based on a Time Range via REST API	979
1.6.1.1.4 Retrieving the Total Record Count of a Table via REST API	981
1.6.1.1.5 Retrieving All Records Matching the Given Search Query via REST API	982
1.6.1.1.6 Retrieving the Number of Records Matching the Given Search Query via REST API	983
1.6.1.1.7 Tracking the Indexing Process Completion via REST API	984
1.6.1.1.8 Clear Index Information of a Given Table via REST API	985
1.6.1.1.9 Defining/Updating the Schema of a Table	986
1.6.1.1.10 Retrieving the Schema of a Table via REST API	988
1.6.1.1.11 Drilling Down Through Categories via REST API	989
1.6.1.1.12 Retrieving Specific Records through a Drill Down Search via REST API	990
1.6.1.1.13 Retrieving the Number of Records Matching the Drill Down Criteria via REST API	99
1.6.1.1.14 Retrieving Records Matching the Given Primary Key Combination via REST API	993
1.6.1.1.15 Retrieving All Record Stores via REST API	996
1.6.1.1.16 Retrieving the Record Store of a Given Table via REST API	997
1.6.1.1.17 Checking if the Given Records Store Supports Pagination via REST API	998
1.6.1.1.18 Tracking the Indexing Process Completion of a Table via REST API	999
1.6.1.1.19 Retrieving Aggregated Values of Given Records via REST API	100
1.6.1.1.20 Retrieving the Event Count of Range Facets via REST API	100
1.6.1.1.21 Re-indexing the Records of a Given Table via REST API	100
1.6.1.2 CORS Settings for the Analytics REST API	100

1.6.1.3 HTTP Status Codes	100
1.6.2 Default Ports of WSO2 Products	100
1.6.2.1 WSO2 DAS Specific Ports	101
1.6.3 WSO2 Patch Application Process	101
1.6.4 Calling Admin Services from Apps	101
1.6.5 Customizing the Management Console	101
1.6.6 Using Analytics Spark Features in Other WSO2 Products	102
1.6.7 WSO2 DAS Performance Analysis	102
1.6.8 Working with Dashboards	102
1.6.8.1 Adding a Custom Theme for a Dashboard	102
1.7 FAQ	103
1.8 Glossary	103
1.9 Getting Support	103
1.10 Site Map	103
2. Tutorials	103
2.1 Create Event Stream	103
2.2 Persist Event Stream	104
2.3 Simulate Events for a Persisted Event Stream	104
2.4 Search for Records via the Management Console	104

WSO2 Data Analytics Server Documentation

WSO2 Data Analytics Server (DAS) is a lean, fully-open source, complete solution for **aggregating** and **analyzing** data and **presenting** information about business activities. It provides real-time visibility into distributed complex systems, including service-oriented architecture (SOA) processes, transactions and workflows.

Watch the [WSO2 screencast](#) to familiarize yourself with WSO2 DAS. This is a great place for you to start if you are new to DAS.

	Get started with WSO2 DAS If you are new to using WSO2 Data Analytics Server, follow the steps given below to get started:
	Get familiar with WSO2 DAS Understand the basics of the DAS and its architecture.
	Quick Start Guide Download, install and run the DAS in just 10 minutes.
	Try out the Samples Try out the DAS real-life business use cases.

For additional learning resources such as webinars and white papers, go to <http://wso2.com/search/resources/>. This is a great place for you to expand your knowledge on WSO2 DAS.

Deep dive into WSO2 DAS

To know more about WSO2 DAS 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.

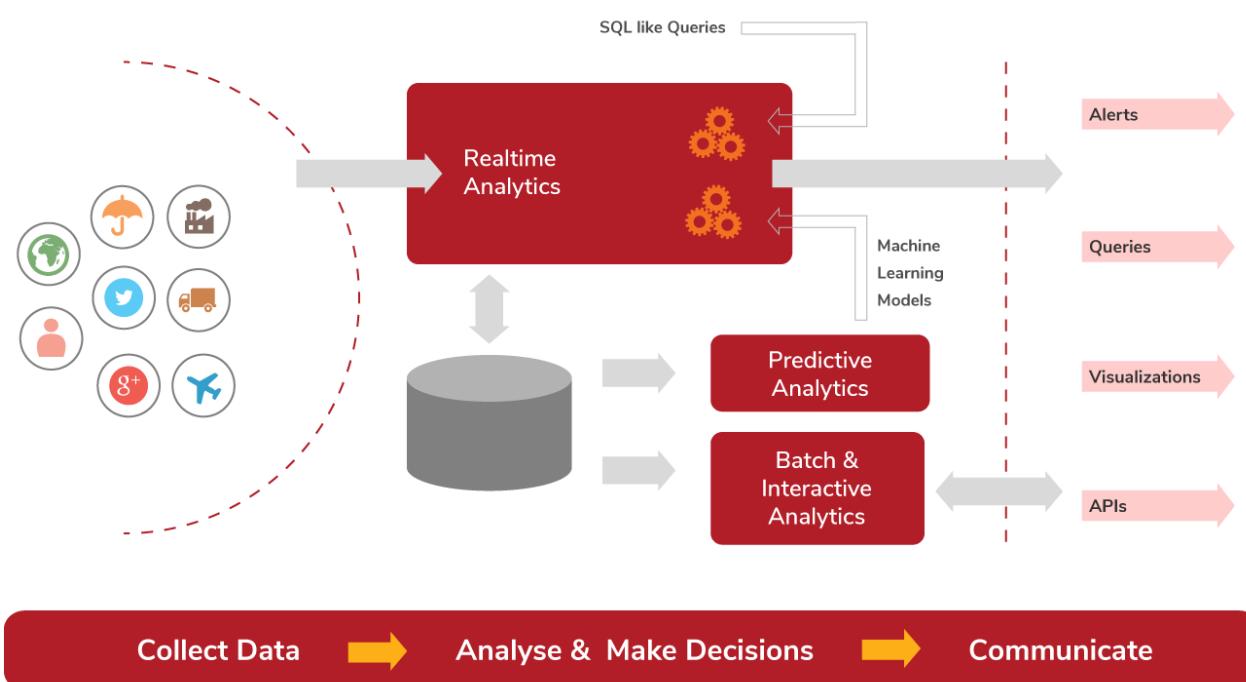
About DAS

The following topics in this section introduce WSO2 Data Analytics Server, including the business cases it solves, its features, and architecture.

- [Introducing DAS](#)
- [DAS Features](#)
- [Architecture](#)
- [About this Release](#)

Introducing DAS

WSO2 Data Analytics Server (DAS) listens to a constant stream of events that represents the transactions and activities of an enterprise from different sources, processes them in real time and communicates the results in a variety of interfaces. This allows organisations to respond quickly to their environments, thus gaining an advantage over their competitors. In addition, WSO2 DAS combines real-time analytics with batch (equipped with incremental processing), interactive and predictive (via machine learning) analysis of data into one integrated platform to support the multiple demands of Internet of Things (IoT) solutions, as well as mobile and Web apps as illustrated by the image below.



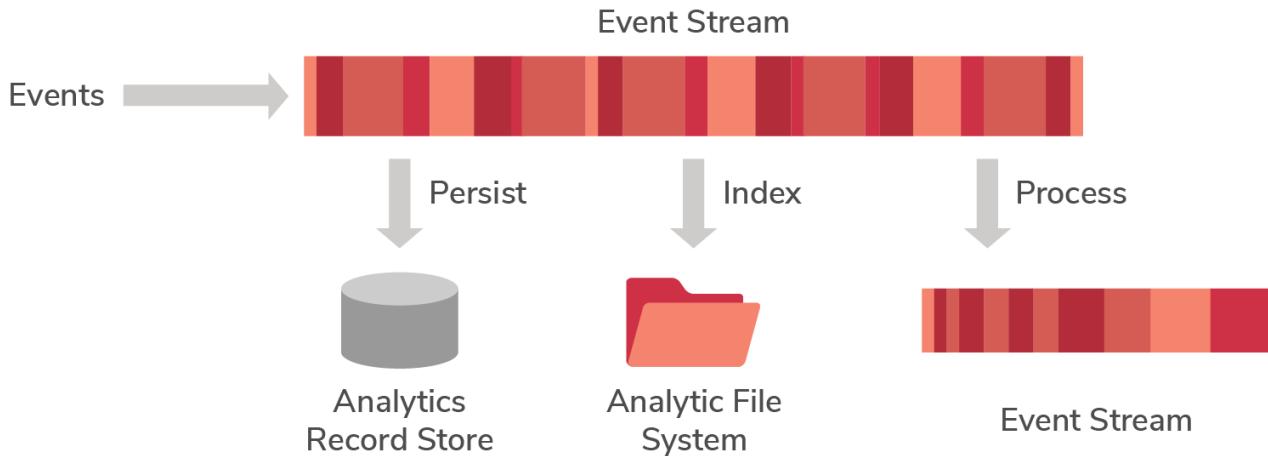
As a part of WSO2's analytics platform, WSO2 DAS introduces a single solution with the ability to build systems and applications that collect and analyze both realtime and persisted, data and communicate the results. It is designed to treat millions of events per second, and is therefore capable to handle Big Data volumes and Internet of Things projects.

WSO2 DAS workflow consists of the following three main phases.

- Collecting Data
- Analyzing Data
- Communicating Results

Collecting data

WSO2 DAS exposes a single API for external data sources to publish data events to it. Further, it provides configurable options either to process the data event stream inflow (in memory) for realtime analytics, to persist (in data storage) for batch analytics and/or to index for interactive analytics as shown in the following diagram.



Creating event streams

The first step in aggregating data is defining an event stream by creating the event stream definition. A stream definition provides the initial structure and identification required for event processing. It includes a set of data types as properties, name, version and other attributes. When an external data publisher sends data events to WSO2 DAS (the receiver), it needs to specify the name and version of the stream intended. Also, data events should be sent according to the structure defined in the stream definition. For more information on event streams, see [Event Streams](#).

Persisting events

This step is required if data needs to be stored in databases for batch analytics. WSO2 DAS introduces a pluggable architecture which allows you to persist data events into any relational data storage (i.e. Oracle, MSSQL, MySQL etc.), or NoSQL storage (i.e. Apache HBase, Apache Cassandra etc.). It is also possible for multi data event storage. For an example, the events can be stored in a NoSQL storage while the processed data events can be stored in a relational data storage.

Creating event receivers

Event Receivers are the connectors to different data sources in WSO2 DAS. WSO2 DAS supports event retrieval from many transport protocols and different formats. For information on the supported transport protocols and event formats, see [Configuring Event Receivers](#).

Analyzing data

You can configure any data event stream received by WSO2 DAS for real time and/or batch analytics.

Realtime analytics

You can process the event streams inflow through the WSO2 real time analytics engine which is powered by Siddhi. The realtime analytic engine can process multiple event streams in realtime. For this, you need to specify a set of queries or rules using the SQL like Siddhi Query Language in an execution plan. Execution plan is the editor for the event processing logic. An execution plan consists of a set of queries and import and export streams. For more information see the following sections.

- [Creating a Standalone Execution Plan](#)
- [Creating a STORM Based Distributed Execution Plan](#)

Batch and interactive analytics

You can perform batch analytics when you configure and persist event streams for batch processing scenarios such as data aggregation, summarization etc. WSO2 DAS batch analytics engine is powered by Apache Spark, which accesses the underlying data storage and executes programs to process the event data. An SQL-like query language is provided to create the jobs that needs to be executed. For more information, see [Batch Analytics Using Spark SQL](#).

Interactive analytics are used when you need to get fast results through adhoc querying of a data set. Interactive analytics in WSO2 DAS is possible when you select to index event stream attributes.

Spark console

You can obtain faster results by executing adhoc queries on the indexed attributes through an interactive Web console, named as the [Batch Analytics Console](#).

Event publishers

Event publishers provide the capability to send event notifications and alerts from WSO2 DAS to external systems. For more information, see [Creating Alerts](#).

Event flow

You can use the [Event Flow](#) feature of WSO2 DAS to visualize how the components of it are connected with each other. Also, you can use it to validate the flow of the events within the DAS.

Event simulation

Event Simulator is a tool which you can use for monitoring and debugging event streams. You need to create event(s) by assigning values to event stream attributes to simulate them. For more information, see [Publishing Data Using Event Simulation](#).

Data Explorer

The Data Explorer is the Web console for searching analytical data. Primary key, data range, facet search are some available options for simple analytical record searches. It is also possible to search records by providing Lucene queries for advanced searches. For more information, see [Data Explorer](#).

Predictive analytics

You can perform predictive analytics for data in WSO2 DAS by integrating it with [WSO2 ML](#). This integration allows WSO2 ML to use an Analytics table in WSO2 DAS as a dataset and make predictions/recommendations using that data by applying Machine Learner algorithms.

Communicating results

WSO2 DAS uses several presentation mechanisms to present event notifications and processing results. For more information, see [Communicating Results](#).

Analytics Dashboard

WSO2 DAS provides an Analytics Dashboard for creating customizable dashboards for visualization of analytical data. Dashboard creation is wizard driven, where you can use widgets/gadgets such as line, bar, and arc charts to get data from analytical tables and add them on a structured grid layout to provide an overall view. For more information, see [Analytics Dashboard](#).

DAS Features

For the relevant versions of the applications used as features, see [Compatibility of WSO2 Products](#).

Feature	Description
---------	-------------

Extremely High Performant Processing Engine	<ul style="list-style-type: none"> Process about 100K+ events per second on single-server commodity hardware Powered by WSO2 Siddhi
Data aggregation	<ul style="list-style-type: none"> Receives data from event sources through Java agents (Thrift, Kafka, JMS), JavaScript clients (Web Sockets, REST), to IoT (MQTT), and also from WSO2 Enterprise Service Bus Connectors. Publishes events to one API for real-time, batch or interactive processing. Ability to access the analytics service via comprehensive REST API.
Powerful and Extensible Query Language for Temporal Event Stream Processing	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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
Supports Multiple Alert Notifying Mechanisms	<ul style="list-style-type: none"> 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
Easily Integrates with any Enterprise System for Event Capture	<ul style="list-style-type: none"> 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

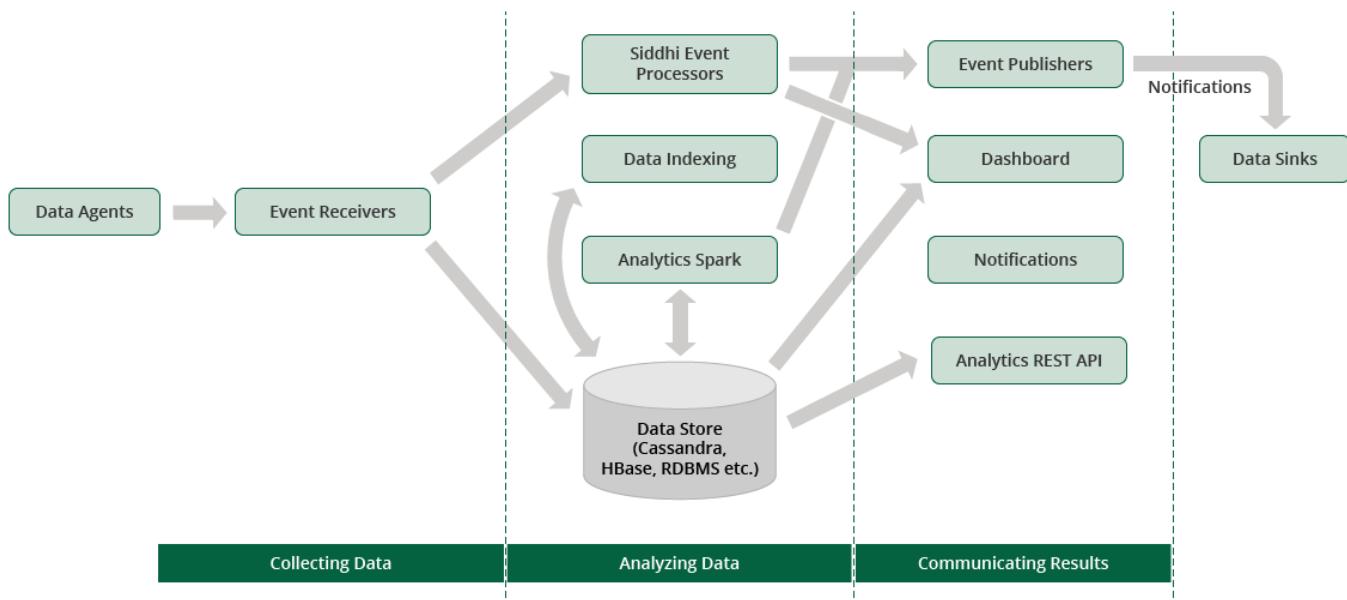
IoT (Internet of Things) Integration	<ul style="list-style-type: none"> MQTT protocol
Debugging Support	<ul style="list-style-type: none"> Event tracing, Event Flow visualization and Event Simulation capabilities Siddhi TryIt to try out Siddhi queries
Massively Scalable	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Run DAS in active-passive deployment with zero downtime Coordinates event and state sharing
Support for Long Duration Queries	<ul style="list-style-type: none"> 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
Integrated, real-time, and batch analytics	<ul style="list-style-type: none"> Analyses both persisted and realtime data using a single product. Fast execution of batch programs using Apache Spark. Detects patterns (fraud detection) by correlating events from multiple data sources in real time using the high performing, open source WSO2 CEP engine powered by WSO2 Siddhi.
Interactive analytics and edge analytics	<ul style="list-style-type: none"> Searches for full text, complex query lookup, distributed indexing support using Apache Lucene for interactive analytics. Correlates/filters events at the edge for edge analytics.
High level language and data storage	<ul style="list-style-type: none"> Use of a structured easy to learn SQL-like query language. Develops complex real-time queries using SQL-like Siddhi query language. Scalable analytic querying using Spark SQL. Support for RDBMS (MSSQL, Oracle, MySQL) as data storages for low to medium scale enterprise deployments. Support for HBase and Cassandra as NoSQL storage for Big Data enterprise deployments.
Extensibility using C-Apps	<ul style="list-style-type: none"> Industry/domain-specific toolboxes to extend the product for business use cases such as fraud detection, GIS data monitoring, activity monitoring etc. Ability to install C-Apps for each WSO2 middleware product, including the analytics functionality available with WSO2 API Manager.
Built-in Support for WSO2 Products	<ul style="list-style-type: none"> Pre-built event sources from other WSO2 products

Communication	<ul style="list-style-type: none"> Possibility to create custom dashboards and gadgets that provide an at-a-glance view as well as a detail view. Detects conditions and generate realtime alerts and notifications (email, SMS, push notifications, physical sensor alarms etc.) Exposes event tables as an API via WSO2 API Manager and WSO2 Data Services Server.
Manage and Monitor	<ul style="list-style-type: none"> 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

Data analytics refer to aggregating, analyzing and presenting information about business activities. This definition is paramount, when designing a solution to address a data analysis use case. Aggregation refers to the collection of data, analysis refers to the manipulation of data to extract information, and presentation refers to representing this data visually or in other ways such as alerts. Data which you need to monitor or process sequentially go through these modules.

The WSO2 DAS architecture reflects this natural flow in its very design as illustrated below.



The WSO2 DAS architecture consists of the following components as described below.

- Data Agents
- Event Receivers
- Analytics REST API
- Data Store
- Siddhi Event Processors
- Analytics Spark
- Data Indexing
- Event Publishers

- Analytics Dashboard
- Event Sinks

Data Agents

Data Agents are external sources that publish data to WSO2 DAS. Data Agent components reside in external systems and push data as events to the DAS. For more information on Data Agents, see [Data Agents](#).

Event Receivers

WSO2 DAS receives data published by Data Agents through Event Receivers. Each event receiver is associated with an event stream, which you then persist and/or process using [Event Processors](#). There are many transport types supported as entry protocols for Event Receivers in WSO2 DAS. For more information on Event Receivers, see [Configuring Event Receivers](#).

Analytics REST API

In addition to the Event Receivers, WSO2 DAS facilitates REST API based data publishing. You can use the REST API with Web applications and Web services. For more information on REST API, see [Analytics REST API Guide](#).

Data Store

WSO2 DAS supports Data Stores (Cassandra, HBase, and RDBMS etc.) for data persistence. There are three main types of Data Stores, namely Event Store, Processed Event Store, and File store. Event Store is used to store events that are published directly to the DAS. Processed Event Store is used to store resulting data processed using [Apache Spark](#) analytics. File store is used for storing [Apache Lucene](#) indexes. For more information on Data Stores, see [DAS Data Access Layer](#).

Siddhi Event Processors

WSO2 DAS uses a realtime event processing engine which is based on Siddhi. For more information on realtime analytics using Siddhi, see [Realtime Analytics Using Siddhi](#).

Analytics Spark

[Apache Spark](#) is used to perform batch analytics operations on the data stored in Event Stores using analytics scripts written in Spark SQL. For more information on Spark analytics, see [Batch Analytics Using Spark SQL](#).

Data Indexing

Data Indexing is a periodically running process which updates the Lucene indexes for the indexed fields in the Event Store configurations of an event stream.

Event Publishers

Output data either from Spark scripts or the Siddhi CEP engine are published from the DAS using event publishers. Event Processors support various transport protocols. For more information on Event Publishers, see [Creating Alerts](#).

Analytics Dashboard

Analytics Dashboard is used for data visualization in WSO2 DAS. It consists of several dashboards each with a set of gadgets. You can use either data from [Data Store](#) or from a realtime event stream as the source of data for each gadget.

Event Sinks

Event sinks are the components outside the DAS. Event Publishers send various event notifications to Event Sinks. For more information on Event Publishers, see [Creating Alerts](#).

WSO2 DAS event flow

The event flow of WSO2 DAS is as follows.

1. Event Receivers and the analytics REST API send data to the DAS server.
2. Received data are stored through the data layer in the underlying Data Store (Cassandra, RDBMS or HBase etc.).
3. A background data indexing process fetches the data from the Data Store, and does the indexing operations.
4. Analyzer engine, which is powered by Apache Spark or the realtime Siddhi based Event Processors analyze this data according to defined analytic queries. This usually follows a pattern of retrieving data from the Data Store, performing a data operation such as an addition, and storing data back in the Data Store.
5. The Analytics Dashboard queries the Data Store for the analyzed data and displays them graphically.

About this Release

WSO2 DAS version 3.1.0 is the successor of version 3.0.1. WSO2 DAS 3.1.0. It contains the following new features and enhancements.

What is new in this release

- The integration of WSO2 Machine Learner features.
- Supports **incremental data processing**.
- Improved gadget generation wizard.
- Cross-tenant support.
- Improved CarbonJDBC connector.
- Improvements to facet based aggregations.
- Supports index based sorting
- Supports Spark on YARN for DAS
- Improved indexing functionality.

Compatible WSO2 product versions

You can make any WSO2 product compatible with WSO2 DAS after installing a data agent. Data agents allow the product to communicate with the DAS and send statistics for analysis. The following products have the latest compatible data agents in them by default, and therefore are compatible with DAS 3.1.0.

- APIM 2.0.0.
- AS 5.3.0
- ESB 5.0.0

Fixed issues

For a list of fixed issues in this release, see [WSO2 DAS 3.1.0 - Fixed Issues](#).

Known issues

For a list of known issues in this release, see [WSO2 DAS 3.1.0 - Known Issues](#).

Getting Started

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

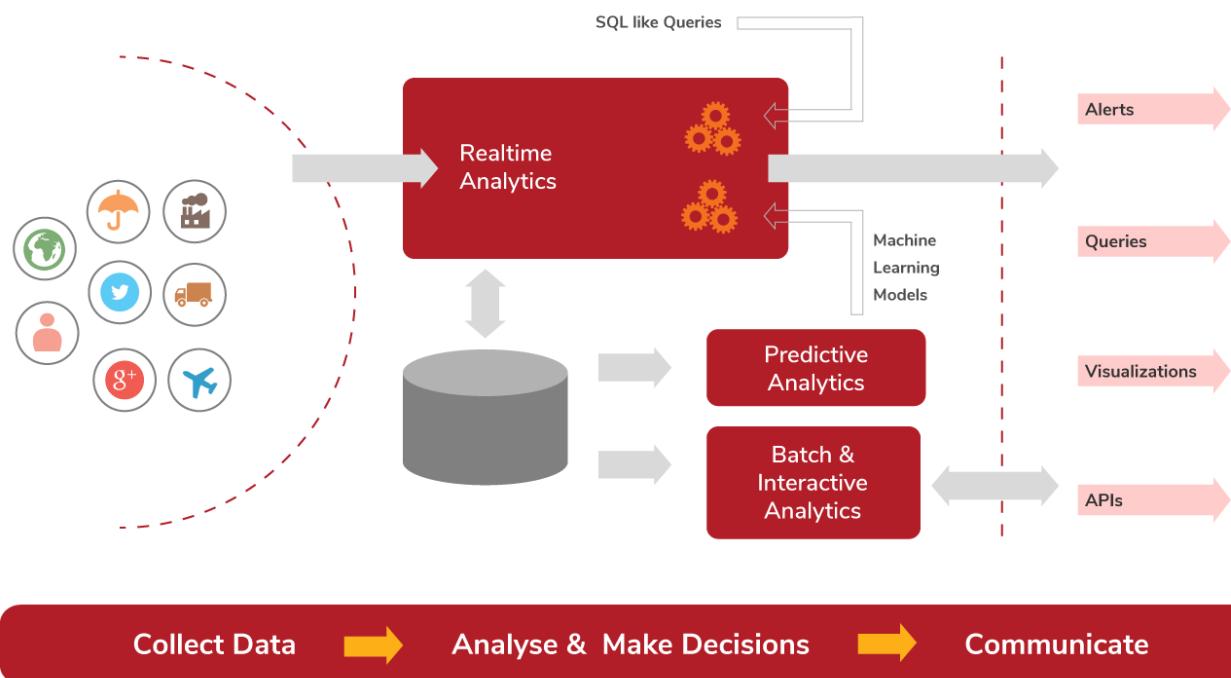
- Quick Start Guide
- Downloading the Product
- Installation Prerequisites
- Installing the Product
- Building from Source
- Upgrading from WSO2 DAS 3.0.1
- Running the Product
- WSO2 DAS Production Setup Checklist

[Go to Home Page](#)

Quick Start Guide

To view a screencast of the Quick Start Guide, click [here](#).

WSO2 Data Analytics Server 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. In addition, it combines real time analytics with batch, interactive, and predictive (via machine learning) analysis of data into one integrated platform to support the multiple demands of Internet of Things (IoT) solutions, as well as mobile and Web apps. It is designed to analyse millions of events per second, and is therefore capable to handle large volumes of data in Big Data and Internet of Things projects. WSO2 DAS workflow consists of three main phases as illustrated in the diagram below.



The following sections walk you through the basic features of the DAS 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
- Deploying the sample C-App
 - Publishing events
 - Viewing the output
- Batch and interactive analytics
 - Step 1: Persist event stream
 - Step 2: Simulate events
 - Step 3: Create and execute Spark scripts
 - Step 4: Search for data
- Predictive Analytics
 - Step 1: Create a dataset
 - Step 2: Create a project
 - Step 3: Create an analysis and train a model
 - Step 4: Predict using the model
- Where to go next

Before you begin,

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

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 its types. An event stream can be created in the DAS Management Console as follows.

1. Log into the DAS 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`.

Home > Manage > Event > Streams

Define New Event Stream

[Help](#)

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	org.wso2.event.sensor.stream <small>Name of the Event Stream</small>
Event Stream Version*	1.0.0 <small>Version of the event stream (Eg : 1.0.0)</small>
Event Stream Description	<small>Description of the event stream</small>
Event Stream Nick-Name	<small>Nick name of the event stream</small>

Stream Attributes

Meta Data Attributes

Attribute Name	Attribute Type	Actions
timestamp	long	Delete
isPowerSaverEnabled	bool	Delete
sensorId	int	Delete
sensorName	string	Delete

Attribute Name : Attribute Type :

Correlation Data Attributes

Attribute Name	Attribute Type	Actions
longitude	double	Delete
latitude	double	Delete

Attribute Name : Attribute Type :

Payload Data Attributes

Attribute Name	Attribute Type	Actions
humidity	float	Delete
sensorValue	double	Delete

Attribute Name : Attribute Type :

Event Stream Details

Parameter Name	Value
Event Stream Name	org.wso2.event.sensor.stream
Event Stream Version	1.0.0

Stream Attributes

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

Attribute Category	Attribute	Attribute Type
Meta Data	timestamp	long

	isPowerSaverEnabled	bool
	sensorId	int
	sensorName	string
Correlation Data	longitude	double
	latitude	double
Payload Data	humidity	float
	sensorValue	double

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

Step 2: Add an event receiver

Events received by the DAS 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 DAS.

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 DAS 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`.

? Help

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name*	<input type="text" value="httpReceiver"/> <small>⑦ Enter a unique name to identify Event Receiver</small>
From	
Input Event Adapter Type*	<input type="text" value="http"/> <small>⑦ Select the type of Adapter to receive events</small>
<small>Following url formats are used to receive events</small> For super tenants: <code>http://localhost:9764/endpoints/<event_receiver_name></code> <code>https://localhost:9444/endpoints/<event_receiver_name></code>	
<small>For other tenants:</small> <code>http://localhost:9764/endpoints/t/<tenant_domain>/<event_receiver_name></code> <code>https://localhost:9444/endpoints/t/<tenant_domain>/<event_receiver_name></code>	
Adapter Properties	
Transport(s)*	<input type="text" value="all"/> <small>⑦</small>
To	
Event Stream*	<input type="text" value="org.wso2.event.sensor.stream:1.0.0"/> <small>⑦ The event stream that will be generated by the received events</small>
Mapping Configuration	
Message Format*	<input type="text" value="json"/> <small>⑦ Select the input message format</small>
+ Advanced	

[Add 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

- 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 DAS CLI in text format. The publisher can be created using the Management Console as follows.

1. Log into the DAS 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.

The screenshot shows the 'Create a New Event Publisher' form. The 'Event Publisher Name' field contains 'loggerPublisher'. The 'Event Source' dropdown is set to 'org.wso2.event.sensor.stream:1.0.0'. The 'Stream Attributes' section displays a list of attributes: meta_timestamp long, meta_isPowerSaverEnabled bool, meta_sensorId int, meta_sensorName string, correlation_longitude double, correlation_latitude double, humidity float, sensorValue double. The 'Output Event Adapter Type' dropdown is set to 'logger'. The 'Unique Identifier' field is empty. The 'Message Format' dropdown is set to 'text'. At the bottom, there are 'Add Event Publisher' and 'Test Event Publisher' buttons.

Parameter Name	Description
Event Publisher Name	loggerPublisher
Event Source	org.wso2.event.sensor.stream
Output Event Adapter Type	logger
Message Format	text

- 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.

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



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

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

This builds the HTTP client and sends the events in the `<DAS_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](#).

```
[echo] Configure -Durl=xxxx and (-DfilePath=xxxx or -Dsn='sample number') optionally use -Dusername=xxxx -Dpassword=xxxx
[java] Starting WSO2 Http Client
[java] Sending message:
[java]
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 701,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
[java] Sending message:
[java]
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 702,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
[java] Sending message:
[java]
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 703,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
```

The logs of the JSON events received by the DAS server will be displayed in the CLI as shown in the example below.

```
[2015-05-15 11:00:58,233] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:4354643,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:702,
  meta_sensorName:temperature,
  correlation_longitude:4.504343,
  correlation_latitude:20.44345,
  humidity:2.3,
  sensorValue:4.504343
[2015-05-15 11:00:58,234] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:4354643,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:703,
  meta_sensorName:temperature,
  correlation_longitude:4.504343,
  correlation_latitude:20.44345,
  humidity:2.3,
  sensorValue:4.504343
[2015-05-15 11:00:58,234] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:4354643,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:701,
  meta_sensorName:temperature,
  correlation_longitude:4.504343,
  correlation_latitude:20.44345,
  humidity:2.3,
  sensorValue:4.504343
```

Processing events with an execution plan

Step 6: Add another event stream

1. Log into the DAS 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`.

Home > Manage > Event > Streams
Define New Event Stream

[Help](#)

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	org.wso2.event.sensor.filtered.stream <small>Name of the Event Stream</small>
Event Stream Version*	1.0.0 <small>Version of the event stream (Eg : 1.0.0)</small>
Event Stream Description	 <small>Description of the event stream</small>
Event Stream Nick-Name	 <small>Nick name of the event stream</small>

Stream Attributes

Meta Data Attributes

Attribute Name	Attribute Type	Actions
timestamp	long	Delete
sensorName	string	Delete

Attribute Name : Attribute Type :

Correlation Data Attributes

Attribute Name	Attribute Type	Actions
longitude	double	Delete
latitude	double	Delete

Attribute Name : Attribute Type :

Payload Data Attributes

Attribute Name	Attribute Type	Actions
sensorValue	double	Delete

Attribute Name : Attribute Type :

Event Stream Details

Paramater Name	Value
Event Stream Name	org.wso2.event.sensor.filtered.stream
Event Stream Version	1.0.0

Stream Attributes

Attribute Category	Attribute	Attribute Type
Meta Date	timestamp	long
	sensorName	string
Correlation Data	longitude	double
	latitude	double

Payload Data	sensorValue	double
--------------	-------------	--------

- 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](#).

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

Parameter Name	Value
Import Stream	org.wso2.event.sensor.stream:1.0.0
As	sensorStream
Value Of	filteredStream
StreamId	org.wso2.event.sensor.filtered.stream:1.0.0

- Click **Import** and then click **Export**. The section for query expressions will be updated as shown below.

Home > Manage > Streaming Analytics > Execution Plans > Add

[? Help](#)

Create a New Execution Plan

Enter Event Processor Details

Query Expressions

Import Stream*: org.wso2.event.sensor.stream:1.0.0 As: Import

Export Stream Value Of: StreamId: org.wso2.event.sensor.filtered.stream:1.0.0 Export

```

1 /* Enter a unique ExecutionPlan */
2 #Plan:name('ExecutionPlan')
3
4 /* Enter a unique description for ExecutionPlan */
5 -- #Plan:description('ExecutionPlan')
6
7 /* define streams/tables and write queries here ... */
8
9 #Import('org.wso2.event.sensor.stream:1.0.0')
10 define stream sensorStream (meta_timestamp long, meta_isPowerSaverEnabled bool,
11 meta_sensorId int, meta_sensorName string, correlation_longitude double,
12 correlation_latitude double, humidity float, sensorValue double);
13
14 #Export('org.wso2.event.sensor.filtered.stream:1.0.0')
15 define stream filteredStream (meta_timestamp long, meta_sensorName string,
16 correlation_longitude double, correlation_latitude double, sensorValue double);
17

```

- Add the following query expression.

Enter Event Processor Details

Query Expressions

Import Stream*: org.wso2.event.sensor.stream:1.0.0 As: Import

Export Stream Value Of: StreamId: org.wso2.event.sensor.filtered.stream:1.0.0 Export

```

1 /* Enter a unique ExecutionPlan */
2 @Plan:name('ExecutionPlan')
3
4 /* Enter a unique description for ExecutionPlan */
5 == @Plan:description('ExecutionPlan')
6
7 /* define streams/tables and write queries here ... */
8
9 @Import('org.wso2.event.sensor.stream:1.0.0')
10 define stream sensorStream (meta_timestamp long, meta_isPowerSaverEnabled bool,
11 meta_sensorId int, meta_sensorName string, correlation_longitude double,
12 correlation_latitude double, humidity float, sensorValue double);
13
14 @Export('org.wso2.event.sensor.filtered.stream:1.0.0')
15 define stream filteredStream (meta_timestamp long, meta_sensorName string,
16 correlation_longitude double, correlation_latitude double, sensorValue double);
17
18 from sensorStream [sensorValue > 100]
19 select meta_timestamp, meta_sensorName, correlation_longitude,
correlation_latitude, sensorValue
20 insert into filteredStream

```

```

from sensorStream [sensorValue > 100]
    select meta_timestamp, meta_sensorName, correlation_longitude,
correlation_latitude, sensorValue
        insert into filteredStream

```

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 DAS 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`.

? Help

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name* uiPublisher <small>© Enter a unique name to identify Event Publisher</small>	From Event Source* <small>org.wso2.event.sensor.filtered.stream:1.0.0</small> <small>© The stream of events that need to be published</small>
Stream Attributes <small>meta_timestamp long, meta_sensorName string, correlation_longitude double, correlation_latitude double, sensorValue double</small>	
To Output Event Adapter Type* <small>ui</small> <small>© Select the type of Adapter to publish events</small>	
Usage Tips <small>There must be an UI output adaptor for each stream to be visualized via Analytics Dashboard.</small>	
Mapping Configuration Message Format* <small>wso2event</small> <small>© Select the output message format</small>	
Advanced	

[Add Event Publisher](#)
[Test Event Publisher](#)

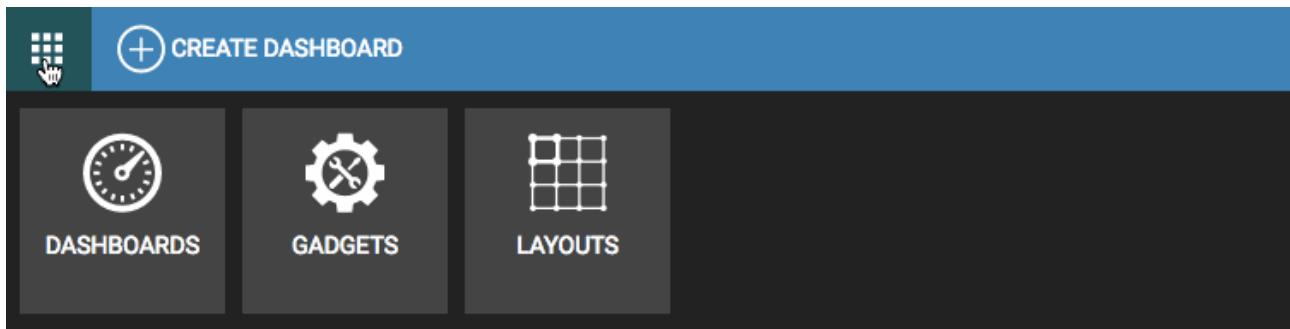
Parameter Name	Description
Event Publisher Name	uiPublisher
Event Source	org.wso2.event.sensor.filtered.stream:1.0.0
Output Event Adapter Type	ui
Message Format	wso2event

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 DAS 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.



Dashboards

A dark blue card for a 'Geo Dashboard'. It displays the title 'Geo Dashboard', a URL 'URL: geo-dashboard', and three icons for 'View', 'Design', and 'Settings'.

4. Click **GENERATE GADGET**, and enter values in the **Generate a Gadget** wizard as follow.

Generate a Gadget

The screenshot shows a three-step wizard for generating a gadget. Step 1, 'Select Provider', is active and highlighted in blue. Step 2, 'Configure Provider', and Step 3, 'Configure Chart', are shown below it. On the left is a 'Previous' button with a left arrow icon, and on the right is a 'Next' button with a right arrow icon. The provider selection dropdown is open, showing three options: 'Relational Database Source' (selected, indicated by a checked checkbox and a blue background), 'Realtime Data Source', and 'REST Data Source'. A cursor arrow points towards the 'Realtime Data Source' option.

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

Parameter Name	Value
Gadget Name	Sensor Value VS Timestamp
Select Chart Type	Line Chart
X-Axis	TIMESTAMP
X type	time
Y-Axis	sensorValue
Y type	default
Color domain	sensorName
Max length	30

- d. 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.

The screenshot shows the 'Dashboards' page of the WSO2 Data Analytics Server. At the top, there's a blue header bar with a 'CREATE DASHBOARD' button containing a plus sign and a hand cursor icon. Below the header, the main content area has a light gray background. It displays the message 'No dashboards found.' in a small font, followed by 'To create a dashboard click [here](#)' in a smaller font.

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

Parameter Name	Value
Name of your Dashboard	Sensor Statistics
Description	This dashboard indicates the sensor value at different times in a particular location.

- b. Select the **Single Column** layout. A message appears to indicate that the dashboard is successfully created.
- c. 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 DAS CLI by the [publisher created in Step 3](#).

1. Issue the following curl command.

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

The following log will appear in the DAS CLI.

```
[2015-08-13 16:52:42,033] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -\nUnique ID: loggerPublisher,\nEvent: meta_timestamp:1439468145264,\nmeta_isPowerSaverEnabled:false,\nmeta_sensorId:701,\nmeta_sensorName:temperature,\ncorrelation_longitude:4.504343,\ncorrelation_latitude:20.44345,\nhumidity:2.3,\nsensorValue:96.5
```

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.

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

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

The following log will appear in the DAS CLI.

```
[2015-08-13 17:36:21,463] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -\nUnique ID: loggerPublisher,\nEvent: meta_timestamp:1439467524120,\nmeta_isPowerSaverEnabled:false,\nmeta_sensorId:701,\nmeta_sensorName:temperature,\ncorrelation_longitude:4.504343,\ncorrelation_latitude:20.44345,\nhumidity:2.3,\nsensorValue:156.0
```

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.

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

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

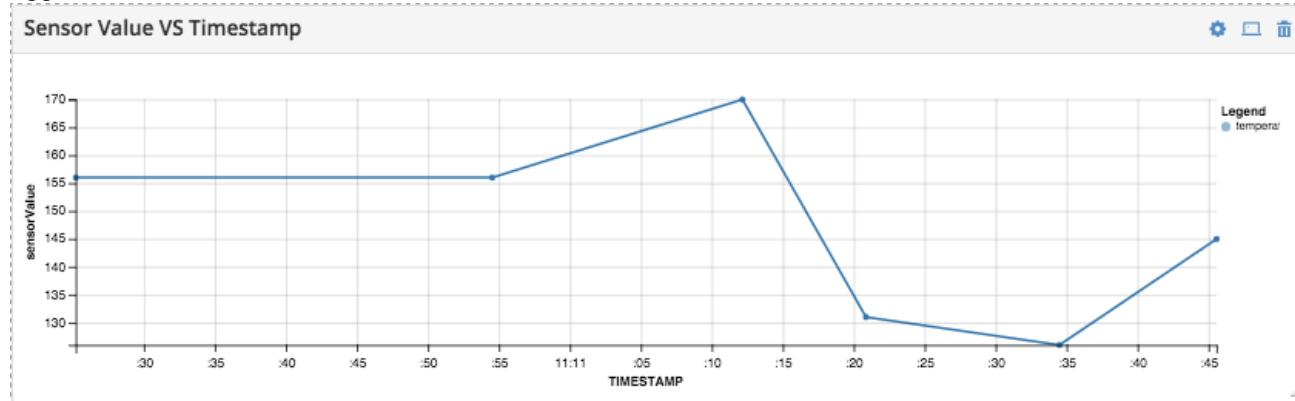
```
curl -X POST -d "{ \"event\": { \"metaData\": { \"timestamp\": 1439467951518 ,\n\"isPowerSaverEnabled\": false, \"sensorId\": 701, \"sensorName\": temperature },\n\"correlationData\": { \"longitude\": 4.504343, \"latitude\": 20.44345 },\n\"payloadData\": { \"humidity\": 2.3, \"sensorValue\": 131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

```
curl -X POST -d "{ \"event\": { \"metaData\": { \"timestamp\": 1439467992936 ,\n\"isPowerSaverEnabled\": false, \"sensorId\": 701, \"sensorName\": temperature },\n\"correlationData\": { \"longitude\": 4.504343, \"latitude\": 20.44345 },\n\"payloadData\": { \"humidity\": 2.3, \"sensorValue\": 126 } } }\"\nhttp://localhost:9763/endpoints/httpReceiver --header\n\"Content-Type:application/json"
```

```
curl -X POST -d "{ \"event\": { \"metaData\": { \"timestamp\": 1439468050928 ,\n\"isPowerSaverEnabled\": false, \"sensorId\": 701, \"sensorName\": temperature },\n\"correlationData\": { \"longitude\": 4.504343, \"latitude\": 20.44345 },\n\"payloadData\": { \"humidity\": 2.3, \"sensorValue\": 145 } } }\"\nhttp://localhost:9763/endpoints/httpReceiver --header\n\"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 DAS 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 DAS CLI as well.



Deploying execution plans using templates

Step 11: Create and deploy a template

In this step, the configurations of the WSO2 DAS 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 `<DAS_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 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
~
~     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.
-->

<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"
                            }
                        ],
                        "payloadData": [
                            {
                                "name": "humidity",
                                "type": "FLOAT"
                            },
                            {
                                "name": "sensorValue",
                                "type": "DOUBLE"
                            }
                        ]
                    }
                
            
        
    


```

```

        }
    ]
}
</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>
<template type="eventreceiver">
<![CDATA[

        <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[

        <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>
<template type="eventpublisher">
<![CDATA[

        <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" : "$sensorType-line-chart",
    "chart-name" : "line-chart"
}
}
]]>
</artifact>
<artifact file="js/core/gadget-util.js">
<![CDATA[
var getGadgetLocation = function (callback) {
    var gadgetLocation =
"/portal/store/carbon.super/fs/gadget/$sensorType-line-chart";
    var PATH_SEPERATOR = "/";
    if (gadgetLocation.search("store") != -1) {

wso2.gadgets.identity.getTenantDomain(function (tenantDomain) {
    var gadgetPath =
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);
}
}
]]>

```

```

        ]]>
    </artifact>
</artifacts>
</config>
</template>
<!-- Gadget line chart -->
<!-- Dashboard -->
<template type="dashboard">
    <config>
        <properties>
            <property
name="dashboardId">analytics-$sensorType-dashboard</property>
        </properties>
        <content>
<![CDATA[
    {
        "id": "analytics-$sensorType-dashboard",
        "title": "Analytics $sensorType Dashboard",
        "description": "This dashboard indicates the sensor value at different times
in a particular location",
        "permissions": {
            "viewers": [
                "Internal\\sensor-statistics-viewer"
            ],
            "editors": [
                "Internal\\sensor-statistics-editor"
            ],
            "owners": [
                "Internal\\sensor-statistics-owner"
            ]
        },
        "pages": [
            {
                "id": "landing",
                "title": "Home",
                "layout": {
                    "content": {
                        "loggedIn": {
                            "blocks": [
                                {
                                    "id": "90dfe9100dc10dc1ae562e7f7451a4a",
                                    "x": 0,
                                    "y": 0,
                                    "width": 12,
                                    "height": 3,
                                    "banner": false
                                }
                            ]
                        }
                    }
                },
                "fluidLayout": false
            },
            "isanon": false,
            "content": {
                "default": {
                    "90dfe9100dc10dc1ae562e7f7451a4a": [
                        {
                            "id": "$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",
            "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>
```

```

    </scenario>
  </scenarios>
</domain>

```

Step 12: Configure a template

Before you carry out this step

1. Copy Sensor Value VS Timestamp gadget which resides in the <DAS_HOME>/repository/deployment/server/jaggeryapps/portal/store/carbon.super/fs/gadget/ directory and copy the directory to wso2das-3.1.0/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 DAS server was running when you [created and deployed the template](#), restart the DAS server.
2. Log into the DAS Management Console. Click the **Main** tab and then click **Template Manager**. Create a new scenario as demonstrated below.

Domains

Select a Domain to proceed

- Click on **SensorStatistics** to open the **Deployed Scenarios** page. Then click **Create New Scenario** to open the **Edit Scenario** page.
- Enter information as shown in the table below and click **Add scenario**.

Parameter Name	Value
Scenario Type	AnalyzeSensorStatistics
Scenario Name	FilterSensorValues
Description	Filter events with a sensor value greater than 120
Sensor Value	120

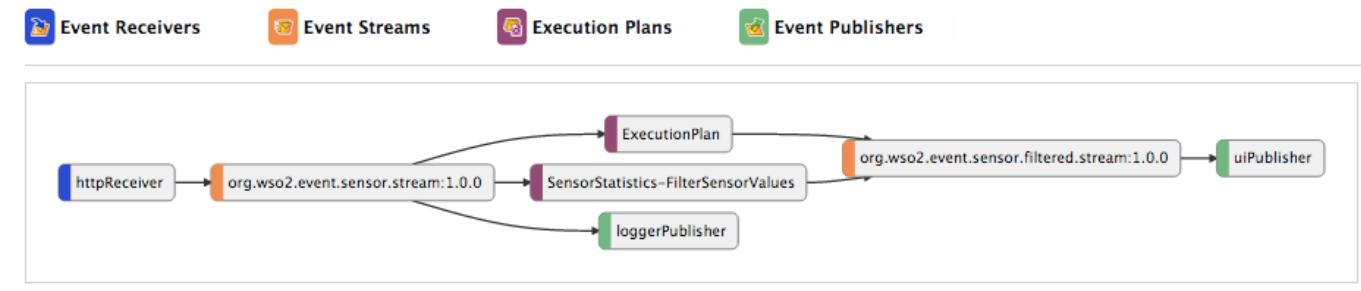
A message appears to inform you that the scenario is successfully created. Close the message. The scenario you configured is displayed in the **Deployed Scenarios** page

Step 13: View the elements of the event flow

Log into the DAS Management Console and click the **Main** tab. Then click **Flow** to open the **DAS 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.

CEP Event Flow



The following is a summary of this guide which describes each element in the event flow.

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

Deploying the sample C-App

You can deploy artifacts (i.e. event streams, event receivers, Spark scripts, event publishers, and dashboards etc.) as composite Carbon Applications (C-Apps) in WSO2 DAS. This guide uses the `SMART_HOME.car` file as the toolbox which contains all the artifacts required for this guide in a single package. For more information on C-Apps, see [Packaging Artifacts as a C-App Archive](#). Follow the steps below to deploy and use a sample C-App in WSO2 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>/capps/Smart_Home.car` file as shown below.

Home > Manage > Carbon Applications > Add

Add Carbon Applications

Upload Carbon Application

Carbon Application Artifact(.car) Smart_Home.car

4. Click **Main**, then click **Carbon Applications**, and then click **List view**, to see the uploaded Carbon application as shown below.

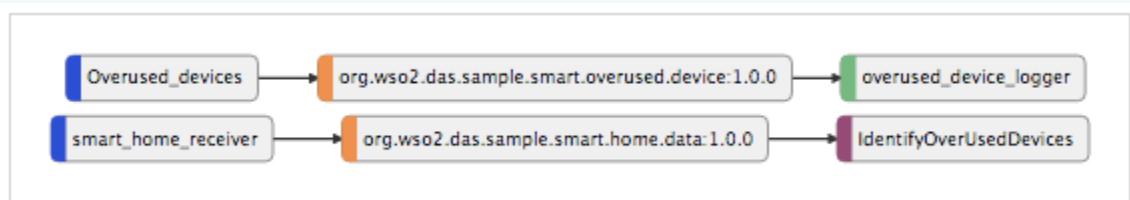
Home > Manage > Carbon Applications > List

Carbon Applications List

1 Running Carbon Applications.

Carbon Applications	Version	Actions
Smart_Home_Sample_CApp	1.0.0	Delete Download

You can use the **Event Flow** feature of WSO2 DAS to visualize how the components that you created above are connected with each other. Also, you can use it for verification purposes i.e. to validate the flow of the events within the DAS as shown below.



Publishing events

Once you develop the complete Event Flow, you can test the flow by publishing the events to the DAS. There are several methods of publishing to DAS. In this section, the events are published via a log file.

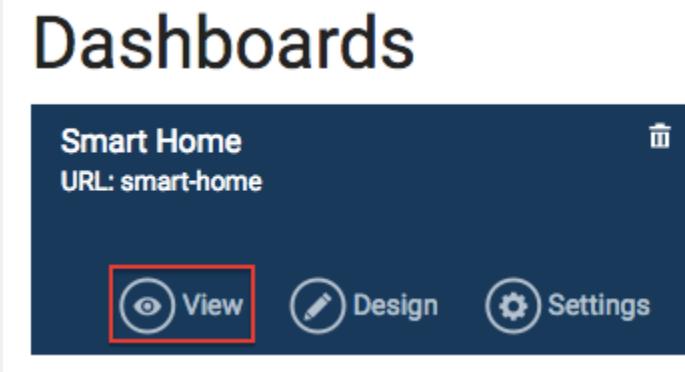
Navigate to `<DAS_HOME>/samples/smart-home/` directory in a new CLI tab, and execute the following command to run the data publisher: ant

This executes a Java client based on the `<DAS_HOME>/samples/smart-home/src/main/java/org/wso2/carbon/das/smarthome/sample/SmartHomeAgent.java` file. This Java client generates random events and sends them to the event stream that is deployed through the `Smart_Home.car` file.

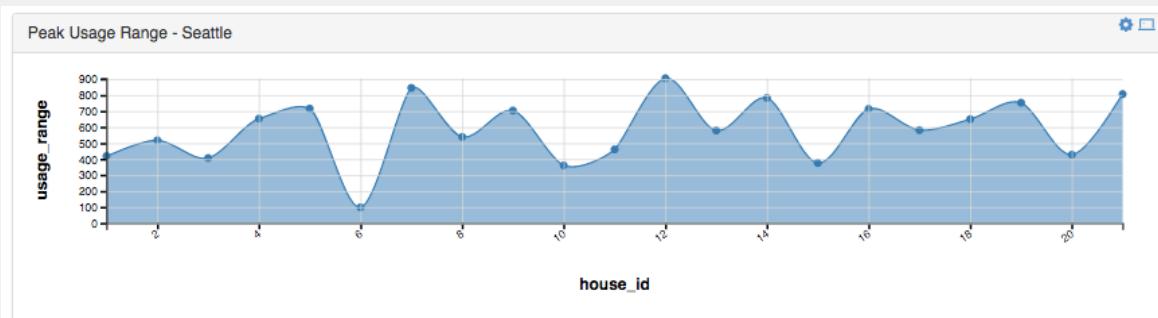
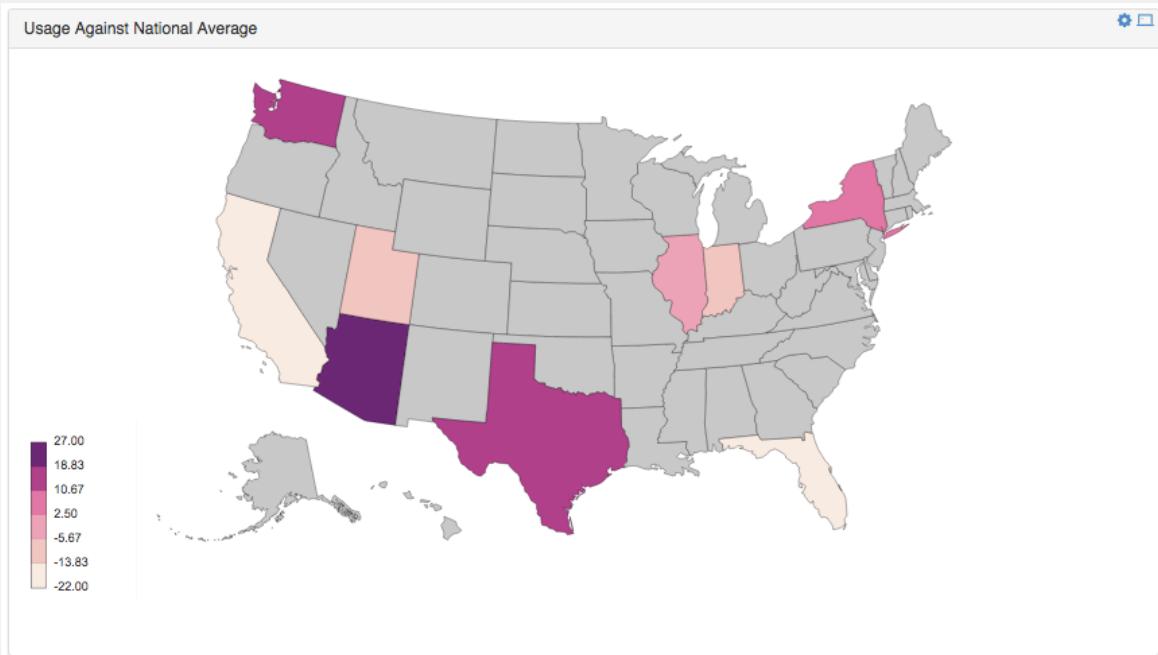
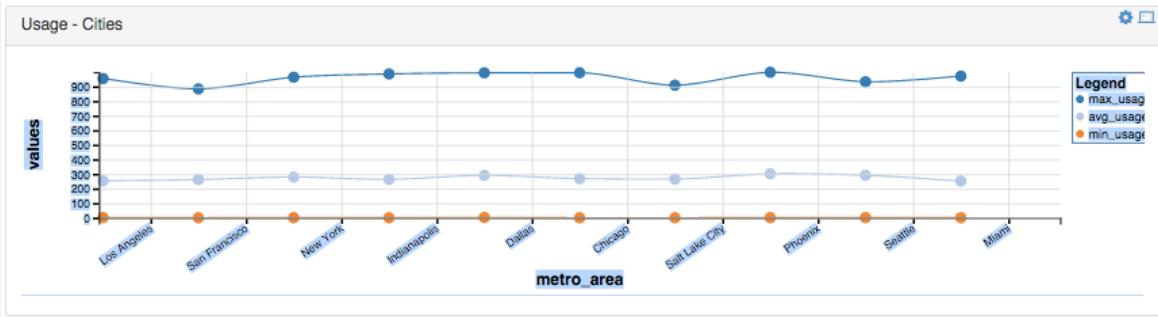
Viewing the output

Follow the steps below to view the presentation of the output in the Analytics Dashboard.

1. Log in to the Management console, if you are not already logged in.
2. Click **Main**, and then click **Analytics Dashboard** in the **Dashboard** menu.
3. Log in to the Analytics Dashboard, using admin/admin credentials.
4. Click the **DASHBOARDS** button in the top menu. The dashboard deployed by the C-App is displayed as shown below.



5. Click the **View** button of the corresponding Dashboard. The following charts are displayed.



Follow the steps below to undeploy the C-App, which you already uploaded in this section before proceeding to the next sections.

1. Log in to the DAS Management Console using admin/admin credentials, if you are not already logged in.
2. Click **Main**, then click **Carbon Applications**, and then click **List view**, to see the uploaded Carbon application.
3. Click on the **Delete** option to delete the Carbon application as shown below.

Carbon Applications	Version	Actions
Smart_Home	1.0.0	Delete Download

4. Refresh the Web browser screen, and check if the SMART_HOME.car file has been removed from the list of all available C-Apps.

Batch and interactive analytics

You can perform batch analytics when event streams are configured to be persisted for later batch processing scenarios such as data aggregation, summarization etc. WSO2 DAS batch analytics engine is powered by Apache Spark, which accesses the underlying data storage and executes programs to process the event data. The DAS provides an SQL-like query language to create the jobs through scripts that need to be executed.

Step 1: Persist event stream

In this step, the `org.wso2.event.sensor.stream` and `org.wso2.event.sensor.filtered.stream` event streams that you previously created are persisted so that the data received by them are stored in the databases configured for WSO2 DAS by default.

The screenshot shows the 'Edit Event Stream' page with the following configuration:

- Persist Event Stream:** Checked.
- Record Store:** EVENT_STORE.
- Meta Data Attributes:**

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	isPowerSaverEnabled	BOOLEAN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	sensorid	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	sensorName	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
- Correlation Data Attributes:**

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	longitude	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	latitude	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
- Payload Data Attributes:**

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	humidity	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	sensorValue	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
- Arbitrary Data Attributes:** No arbitrary data attributes are defined.
- Advanced:** A section with fields for Attribute Name (text input), Attribute Type (dropdown: INTEGER), Primary Key (checkbox), Index Column (checkbox), Score Param (checkbox), Is a Facet (checkbox), and an Add button.
- Buttons:** Back, Save Event Stream.

1. Log in to the WSO2 DAS Management Console if you are not already logged in.
2. In the **Main** tab, click **Streams** to open the **Available Event Streams** page.
3. Click **Edit** for the event stream `org.wso2.event.sensor.stream`. This opens the **Edit Event Stream** page.
4. Click **Next [Persist Event]**.
5. Select the **Persist Event Stream** check box.
6. Select the **Persist Attribute** check box for all the available attributes.
7. Select the **Index Column** check box for the `sensorid` attribute.
8. Click **Save Event Stream** to save the changes.

Step 2: Simulate events

In this step, multiple events are simulated to be stored in the database for batch analytics.

1. Download and save [this file](#) in a preferred location in your machine.

2. Log into the DAS management console using the following URL: `https://<DAS_HOST>:<DAS_PORT>/carbon/`.
3. Click **Tools**, and then click **Event Simulator**.
4. Select `org.wso2.event.sensor.stream` in the **Event Stream Name** field.

Event Stream Simulator

Send single event

Event Stream Name *	<input type="text" value="org.wso2.event.sensor.stream:1.0.0"/>
Stream Attributes	
Meta Attributes	
<code>timestamp(long)</code> *	
<code>isPowerSaverEnabled(bool)</code> *	
<code>sensorid(int)</code> *	
<code>sensorName(string)</code> *	
Correlation Attributes	
<code>longitude(double)</code> *	
<code>latitude(double)</code> *	
Payload Attributes	
<code>humidity(float)</code> *	
<code>sensorValue(double)</code> *	

Send multiple events

File	Stream Configuration	Delay between events(ms)	Action
No file has been uploaded			
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>		

5. Click **Choose File**, and then browse and upload the CSV file you downloaded and saved. Click **Upload** and refresh the page to view the uploaded file.
6. Click **Configure** to open the **Event Mapping Configuration** dialog box. Enter a comma (,) in the **Field delimiter** field, and click **Configure**.

Event Mapping Configuration

File name	<code>org_wso2_event_sensor_stream_1_1_0.csv</code>
Select the target event stream*	<input type="text" value="org.wso2.event.sensor.stream:1.0.0"/>
Field delimiter*	<input type="text" value=","/>
Delay between events in milliseconds*	<input type="text" value="1000"/>
<input type="button" value="Configure"/>	

7. Click **Play** to start sending the events in the file. Click **OK** in the message that appears to confirm that the system has started sending events from the file. The events sent are logged in the CLI as shown below.

```
[2017-02-17 16:41:48,867] ERROR {org.wso2.carbon.event.output.adapter.ui.UIEventAdapter} - Event dropped at Output Adapter 'uiPublisher' for tenant id '-1234', No clients registered
[2017-02-17 16:41:49,864] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: loggerPublisher,
Event: meta_timestamp:19900813115535,
meta_isPowerSaverEnabled:false,
meta_sensorid:502,
meta_sensorName:temperature,
correlation_longitude:90.34344,
correlation_latitude:32.44345,
humidity:13.2,
sensorValue:130.0
[2017-02-17 16:41:50,866] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: loggerPublisher,
Event: meta_timestamp:19900813115535,
meta_isPowerSaverEnabled:false,
meta_sensorid:503,
meta_sensorName:temperature,
correlation_longitude:90.34344,
correlation_latitude:34.0,
humidity:14.5,
sensorValue:150.0
```

Step 3: Create and execute Spark scripts

In this exercise, a Spark query is written to process the data received by WSO2 DAS and stored in the databases.

1. Log in to the DAS Management Console, if you are not already logged in.
2. Click **Main**, and then click **Scripts** in the **Batch Analytics** menu to open the **Available Analytics Scripts** page.
3. Click **Add New Analytics Script**.

4. Enter BATCH_ANALYTICS_SCRIPT in the **Script Name** parameter. Enter the following Spark SQL script in the **Spark SQL Queries** parameter as shown below.

```

CREATE TEMPORARY TABLE sensorData USING CarbonAnalytics OPTIONS
(tableName "ORG_WSO2_EVENT_SENSOR_STREAM", schema "humidity FLOAT,
sensorValue DOUBLE" );

create temporary table highSensorVal using CarbonAnalytics OPTIONS
(tableName "highSensorVal", schema "humidity FLOAT, sensorValue
DOUBLE" );

insert overwrite table highSensorVal select humidity, sensorValue from
sensorData WHERE sensorValue > 100;

Select * from highSensorVal

```

The above script does the following:

- Loads data from the **DAS Data Access Layer (DAL)**, and registers temporary tables in the Spark environment.
- Performs batch processing by fetching the sensor values greater than 100 from `sensorData` table.
- Writes back to a new DAL table named `highSensorVal`.
- Executes the following query:
`Select * from highSensorVal`

5. Click **Add** to save the script. Click **OK** in the message that appears to confirm that the script was successfully saved.

6. Click **Execute** for the script.

Scripts	Actions
Batch_Analytics_Script	

[Add New Analytics Script](#)

The following is displayed to indicate that the queries were successfully executed.

Script Executor – Batch_Analytics_Script

Analytics Script Results

Query: CREATE TEMPORARY TABLE sensorData USING CarbonAnalytics OPTIONS (tableName "ORG_WSO2_EVENT_SENSOR_STREAM", schema "humidity FLOAT, sensorValue DOUBLE")
Query Executed

Query: create temporary table highSensorVal using CarbonAnalytics OPTIONS (tableName "highSensorVal", schema "humidity FLOAT, sensorValue DOUBLE")
Query Executed

Query: insert overwrite table highSensorVal select humidity, sensorValue from sensorData WHERE sensorValue > 100
Query Executed

You can obtain faster results by executing adhoc queries on the indexed attributes through an interactive Web console named the Interactive Analytics Console. Follow the procedure below to perform a batch analytics operation using the Interactive Analytics Console.

1. Log into the DAS Management Console if you are not already logged in.
2. In the **Main** tab, click **Console** to open the **Interactive Analytics Console**.
3. Enter the following query in the console and press the **Enter** key.

Select * from highSensorVal

The output of the query is displayed as follows.

Home > Manage > Batch Analytics > Console

Welcome to interactive analytics SQL shell
This interactive shell lets you execute Spark SQL commands against a Spark cluster
Initialising Spark client...
SparkSQL> admin
admin: Command Not Found
SparkSQL>
SparkSQL> select * from highSensorVal
Show 10 entries
humidity sensorValue
12.2 120.0
13.2 130.0
14.5 150.0
Showing 1 to 3 of 3 entries
SparkSQL>

Step 4: Search for data

This step involves performing interactive analytics for data received by DAS and stored in the configured databases. Searching is supported by the indexing functionality of DAS powered by Apache Lucene. In this step, we search for a specific record by the `sensorid` attribute. This is possible because the `sensorid` attribute was persisted as an index column when you [persisted the event stream](#).

1. Log into the DAS Management Console if you are not already logged in.
2. In the **Main** tab, click **Data Explorer** to open the **Data Explorer** page.
3. In the **Table Name** parameter, select **ORG_WSO2_EVENT_SENSOR_STREAM**.
4. Select the **By Query** option, and enter `meta_sensorid:501` in the data field displayed.

5. Click **Search**. The following records are displayed in the Results section.

Results									
Total Records: 5 (312 ms)									
ORG_WSO2_EVENT_SENSOR_STREAM									
	meta_timestamp	meta_isPowerSaverEnabled	meta_sensorid	meta_sensorName	correlation_longitude	correlation_latitude	humidity	sensorValue	_timestamp
1	19900813115535	False	501	temperature	90.34344	50.44345	12.2	120.0	2017-02-17 18:08:12 IST
2	19900813115535	False	501	temperature	90.34344	30.44345	3.6	30.0	2017-02-17 18:08:06 IST
3	19900813115535	False	501	temperature	90.34344	50.44345	7.2	50.0	2017-02-17 18:08:09 IST
4	19900813115535	False	501	temperature	90.34344	30.44345			2017-02-17 17:09:11 IST
5	19900813115534	False	501	temperature	90.34344	20.44345	2.3	20.0	2017-02-17 18:08:03 IST

<< < 1 > >> Go to page: 1 Row count: 10 Showing 1-5 of 5

Predictive Analytics

WSO2 DAS allows you to build predictive models that analyse data and make predictions.

In this exercise, a dataset is analyzed and a ML model is trained to predict the possibility of a person suffering with breast cancer when data relating to a set of other bodily characteristics is provided.

The dataset used in this scenario contains 10 features to provide data on bodily characteristics, and a response variable with the following labels.

Label	Prediction
2	This value indicates that the situation is benign.
4	This value indicates that the situation is malignant.

The following is a summary of the procedure to make a prediction via WSO2 DAS.

Step 1: Create a dataset

Step 2: Create a project

Step 3: Create an analysis and train a model

Step 4: Predict using the model

Step 1: Create a dataset

Follow the procedure below to upload the dataset based on which the training model is created.

1. Access the WSO2 Machine Learner wizard embedded within DAS via the `https://<DAS_HOME>:<DAS_PORT/ml/site/home/login.jag` URL, and log in with your credentials.

The screenshot shows two main sections: 'Datasets' and 'Projects'. The 'Datasets' section has a purple header with the title 'Datasets' and a small icon of a bar chart. Below it, a light purple area contains text explaining what a dataset is and how it can be uploaded. A dark purple footer bar displays the message 'You have (0) Datasets' and a button labeled '+ ADD DATASET' with a plus sign icon. The 'Projects' section has a similar purple header with the title 'Projects' and a small icon of a cube. Its light purple area contains text explaining what a project is and how it relates to datasets. A dark purple footer bar displays the message 'You have (0) Projects' and a button labeled '+ ADD PROJECT' with a plus sign icon.

2. Click **ADD DATASET** to open the **Create Dataset** page.

3. In the **Data Source** field, click **Choose File** and browse for the <DAS_HOME>/samples/ml/tuned/naive-bayes/breastCancerWisconsin.csv file. Enter values for the rest of the parameters as shown below.

Create Dataset

Dataset Name  *

Version  *

Description 

Breast cancer data in Wisconsin.

Source Type 

Data Source (max size: 100MB) *

breastCancerWisconsin.csv

Data Format 

Column header available 

Parameter Name	Value
Dataset Name	Breast_Cancer_Dataset
Version	1.0.0
Description	Breast cancer data in Wisconsin
Source Type	File
Data Format	CSV
Column Header Available	Yes

4. Click **CREATE DATASET** to save your changes. The **Datasets** page opens and the dataset you entered is displayed as follows.

The screenshot shows the 'DATASETS' section of the WSO2 Data Analytics Server interface. At the top, there's a header bar with a grid icon and the text 'DATASETS'. Below the header, a button labeled '+ CREATE DATASET' is visible. The main content area displays a dataset entry for 'Breast_Cancer_Dataset', showing '1 version available'. To the right of the dataset entry are four buttons: 'CREATE PROJECT' (with a document icon), 'EXPLORE' (with a magnifying glass icon), and 'DELETE DATASET' (with a trash bin icon). A small circular icon with a question mark is also present.

Step 2: Create a project

Follow the procedure below to create a project for the dataset uploaded.

1. Log into the Machine Learner wizard of WSO2 DAS if you are not already logged in. You can access it with the `https://<DAS_HOME>:<DAS_PORT>/ml/site/home/login.jag` URL, and log in with your credentials.
2. Click **ADD PROJECT**.

The image contains two side-by-side cards. The left card, titled 'Datasets', has a purple header and a dark grey footer. It contains text about datasets and their organization, and says 'You have (1) Datasets' with an '+ ADD DATASET' button. The right card, titled 'Projects', has a purple header and a dark grey footer. It contains text about projects and their purpose, and says 'You have (0) Projects' with an '+ ADD PROJECT' button.

3. In the **Create Project** page, enter information as shown below.

Create Project

Project Name ? *

Description ? *

This project performs predictive analysis on the breast cancer data in Wisconsin.

Dataset ? *

Breast_Cancer_Dataset

Create Project

Parameter Name	Description
Project Name	Breast_Cancer_data_analytics_project
Description	This project performs predictive analysis on the breast cancer data in Wisconsin.
Dataset	Breast_Cancer_Dataset

4. Click **Create Project** to save the information. The project is displayed in the **Projects** page as follows.

The screenshot shows the 'PROJECTS \ ' section of the WSO2 Data Analytics Server interface. A new project, 'Breast_Cancer_data_analytics_project', is listed. The project details are as follows:

- Name:** Breast_Cancer_data_analytics_project
- Created:** 2015-12-13 15:12:27.054
- Status:** No analyses available
- Description:** This project performs predictive analysis on the breast cancer data in Wisconsin.
- Analysis name:** e.g. myanalysis
- Actions:** CREATE ANALYSIS, COMPARE MODELS, and DELETE PROJECT buttons.

Step 3: Create an analysis and train a model

Follow the procedure below to analyse the `Breast_Cancer_Dataset` dataset, and then create a training model based on that analysis.

1. Log into the Machine Learner wizard if you are not already logged in. You can access it with the https://<DAS_HOME>:<DAS_PORT>/ml/site/home/login.jsp URL, and login with your credentials.
2. Click the **You have X projects** link as shown below.

The screenshot shows two main sections: 'Datasets' and 'Projects'. The 'Datasets' section has a purple header with the title 'Datasets' and a small icon. Below it is a white area containing a brief description of what a dataset is and how it can be uploaded. At the bottom is a dark grey footer with the text 'You have (1) Datasets' and a button labeled '+ ADD DATASET'. The 'Projects' section has a similar purple header with the title 'Projects' and a small icon. Below it is a white area containing a brief description of what a project is. At the bottom is a dark grey footer with the text 'You have (1) Projects' (which is highlighted with a red box), and a button labeled '+ ADD PROJECT'.

3. Click on the **Breast_Cancer_data_analytics_project** project to expand it.

4. Enter **breast_cancer_analysis_1** as the analysis name and click **CREATE ANALYSIS**. The following page appears displaying the summary statistics.

The screenshot shows the 'Step 1 Preprocess' tab selected in the navigation bar. The main content area is titled 'Summary statistics for a random sample of the dataset'. It contains a table with 11 rows, each representing a feature from the dataset. The columns are 'Feature', 'Type', 'Summary Statistics', and 'Impute'. The 'Type' column shows that most features are 'CATEGORICAL' (represented by pie charts) except for 'SampleCodeNumber' which is 'NUMERICAL' (represented by a histogram). The 'Impute' column for all features is set to 'DISCARD'. The table includes a search bar at the top right and a footer indicating 'Showing 1 to 10 of 11 entries' with navigation links for 'Previous', '1', '2', and 'Next'.

Feature	Type	Summary Statistics	Impute
BareNuclei	CATEGORICAL		DISCARD
BlandChromati	CATEGORICAL		DISCARD
Class	CATEGORICAL		DISCARD
ClumpThickness	CATEGORICAL		DISCARD
MarginalAdhesion	CATEGORICAL		DISCARD
Mitoses	CATEGORICAL		DISCARD
NormalNucleoli	CATEGORICAL		DISCARD
SampleCodeNumber	NUMERICAL		DISCARD
SingleEpithelialCellSize	CATEGORICAL		DISCARD
UniformityOfCellShape	CATEGORICAL		DISCARD

5. Click **Next** without making any changes to the summary statistics.



The **Explore** view opens. Note that **Parallel Sets** and **Trellis Chart** visualizations are enabled, and **Scatter Plot** and **Cluster Diagram** visualizations are disabled. This is determined by the feature types of the dataset.

6. Click **Next**. The **Algorithms** view is displayed. Enter values as shown below.

PROJECTS \ Breast_Cancer_data_analytics_project \ breast_cancer_analysis_1 CANCEL PREVIOUS NEXT

Step 1 Preprocess Step 2 Explore Step 3 Algorithms Step 4 Parameters Step 5 Model

Algorithm

Algorithm name *

Response variable *

Train data fraction

Parameter	Value
Algorithm name	LOGISTIC REGRESSION L_BFGS
Response variable	Class
Train data fraction	0.7

7. Click **Next**. The **Parameters** view appears. Enter **L2** as the reg type.

PROJECTS \ Breast_Cancer_data_analytics_project \ breast_cancer_analysis_1 CANCEL PREVIOUS NEXT

Step 1 Preprocess Step 2 Explore Step 3 Algorithms Step 4 Parameters Step 5 Model

Parameters

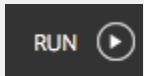
Set Hyper-Parameters for Classification\ LOGISTIC REGRESSION LBFGS

Reg Type ⓘ

8. Click **Next**. The **Model** view appears. Select **Breast_Cancer_Dataset-1.0.0** as the dataset version.

The screenshot shows the 'Step 5 Model' screen of the Machine Learner wizard. At the top, there are navigation buttons: CANCEL, PREVIOUS, RUN, and a right-pointing arrow. Below them, five steps are listed: Step 1 Preprocess, Step 2 Explore, Step 3 Algorithms, Step 4 Parameters, and Step 5 Model. The 'Model' step is highlighted with a dark purple background. The main area is titled 'Model' and contains a 'Dataset version' dropdown menu with the option 'Breast_Cancer_Dataset-1.0.0' selected.

9. Click **RUN** to train the model.



The training model is created as displayed as shown below.

The screenshot shows the 'breast_cancer_analysis_1.Model.2015-12-13_15-15-35' model page. The top bar includes project navigation and a 'BACK' button. The main content area displays the model's name, creation date ('Created: 2015-12-13 15:15:35.503'), and status ('Status: Complete'). It also features 'CREATE MODEL' and 'PUBLISH' buttons. Below the main content, there are 'VIEW', 'PREDICT', and 'DOWNLOAD' links, along with 'DELETE MODEL' and a search bar. A small preview image of the model is visible on the right.

Step 4: Predict using the model

Follow the procedure below to make a prediction based on the training model you created.

1. Log into the Machine Learner wizard if you are not already logged in. You can access it with the URL `https://<DAS_HOME>:<DAS_PORT>/ml/site/home/login.jsp`, and login with your credentials.
2. Click the **You have X projects** link as shown below to open the **Projects** window.

The screenshot shows the 'Projects' window. It has two main sections: 'Datasets' and 'Projects'. The 'Datasets' section contains a brief description and a button to add a dataset. The 'Projects' section contains a brief description and a button to add a project. Both sections show a count of '(1)' datasets and projects respectively. The 'ADD PROJECT' button is highlighted with a red border.

3. Click **MODELS** for the **breast_cancer_analysis_1** analysis.

The screenshot shows the 'Breast_Cancer_data_analytics_project' page. At the top, there is a search bar labeled 'Select a dataset'. Below it is a 'CREATE PROJECT' button. The main content area displays the project details: 'This project performs predictive analysis on the breast cancer data in Wisconsin.' and 'Analysis name: e.g. myanalysis'. There is a 'CREATE ANALYSIS' button. Below this, a list of analyses is shown, with 'breast_cancer_analysis_1' selected. To the right of the analysis list are 'VIEW', 'MODELS' (which is highlighted with a red box), and 'DELETE' buttons.

4. Click **Predict** on the model displayed.

The screenshot shows the 'breast_cancer_analysis_1.Model.2015-12-13_15-15-35' page. The top navigation bar shows 'PROJECTS \ Breast_Cancer_data_analytics_project \ breast_cancer_analysis_1' and a 'BACK' button. The main content area displays the model details: 'breast_cancer_analysis_1.Model.2015-12-13_15-15-35 [Created: 2015-12-13 15:15:35.503]' and '[Dataset version: Breast_Cancer_Dataset-1.0.0] [Status: Complete ✓]'. Below this are 'CREATE MODEL', 'VIEW', 'PREDICT' (which is highlighted with a red box), 'DOWNLOAD', 'PUBLISH', and 'DELETE MODEL' buttons. At the bottom left is a 'Show' dropdown set to '10', and at the bottom right is a page number '1'.

5. Enter values in the **Predict** page as shown below.

Predict

Prediction Source (?)

Feature values

SampleCodeNumber *

1018561

ClumpThickness *

2

UniformityOfCellSize *

1

UniformityOfCellShape *

1

MarginalAdhesion *

1

SingleEpithelialCellSize *

2

BareNuclei *

1

BlandChromati *

1

NormalNucleoli *

1

Mitoses *

5

Predict

Parameter Name	Value
Prediction Source	Feature values
SampleCodeNumber	1018561
ClumpThickness	2
UniformityOfCellSize	1
UniformityOfCellShape	1
MarginalAdhesion	1
SingleEpithelialCellSize	2
BareNuclei	1
BlandChromati	1
NormalNucleoli	1
Mitoses	5

6. Click **Predict**. The prediction is displayed as follows.



Predicted Value:

2

Where to go next

This is your first experience of learning about DAS and trying out its functionalities.

- For more information on the features and architecture of WSO2 DAS, see [About DAS](#).
- For more information on how to download, install, run and get started with WSO2 DAS, see [Getting Started](#).
- For more information on the main functionalities of WSO2 DAS, see [User Guide](#).
- For more information on various product deployment scenarios and other topics useful for system administrators, see [Admin Guide](#).
- For more information on several business use case samples of WSO2 DAS, see [Samples](#).

Downloading the Product

Follow the instructions below to download the binary distribution of WSO2 Data Analytics Server (DAS).

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](#).

1. In your Web browser, go to <http://wso2.com/products/data-analytics-server/>.
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	<ul style="list-style-type: none"> • ~ 8 GB minimum is recommended. • ~ 4 GB heap size.
Disk	<ul style="list-style-type: none"> • ~ 1 GB minimum (excluding space allocated for log files and databases.)

Environment compatibility

Operating Systems / Databases	<ul style="list-style-type: none"> • All WSO2 Carbon-based products are Java applications that can be run on any platform that is Oracle JDK 1.7.*/1.8* compliant. Also, we do not recommend OpenJDK as we do not support it or test our products with it. • All WSO2 products are generally compatible with most common DBMSs. For more information, see Working with Databases. • 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. • For environments that WSO2 products are tested with, see Compatibility of WSO2 Products. • If you have difficulty in setting up any WSO2 product in a specific platform or database, please contact us.
--------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Required applications

The following applications are required for running the Data Analytics Server and its samples, or for building from the source code. Mandatory installs are marked with *.

Application	Purpose	Version	Download Links
-------------	---------	---------	----------------

Oracle Java SE Development Kit (JDK)*	<ul style="list-style-type: none"> • 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. <p>Note</p> <p>To launch WSO2 DAS, you need to have Oracle JDK 1.7.*/1.8.*. You cannot launch WSO2 DAS with Oracle JDK 1.6.* or lower.</p>	1.7 or later / 1.8.*	<p>We do not recommend OpenJDK as we do not support it or test our products with it.</p> <p>http://java.sun.com/javase/downloads/index.jsp</p>
JDBC-compliant Connector for Java	Required as a standardized database driver for Java platforms and development.	1.7.0 or later	http://dev.mysql.com/downloads/connector/
Apache Ant	To compile and run the product samples .	1.7.0 or later	http://ant.apache.org
Git	Required to check out the source from the Git repository.	1.9.0 or later	http://git-scm.com/downloads/

Apache Maven	To build the product from the source distribution (both JDK and Apache Maven are required). <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>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.</p> </div>	3.0.*	http://maven.apache.org
Web Browser	To access each product's Management Console . The Web Browser must be JavaScript enabled to take full advantage of the Management console. <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>WSO2 DAS does not support Internet Explorer 10 and older versions.</p> </div>		

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

- [Installing on Linux](#)
- [Installing on Windows](#)

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 Windows Service](#)
- [Installing as a Linux Service](#)

Installing on Linux

Follow the instructions below to install WSO2 DAS on Linux.

Installing the required applications

1. Establish an SSH connection to the Linux machine or log in on the text Linux console.

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

Installing the DAS

Downloading the Product Download the latest version of the DAS as described in .

2. Extract the archive file to a dedicated directory for the DAS, which will hereafter be referred to as <DAS_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 1.6.0_25 in your system, add the following two lines at the bottom of the file, replacing /usr/java/jdk1.6.0_25 with the actual directory where the JDK is installed.

```
export JAVA_HOME=/usr/java/jdk1.6.0_25
export PATH=${JAVA_HOME}/bin:${PATH}
```

The file should now look like this:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions
export JAVA_HOME=/usr/java/jdk1.6.0_25
export PATH=${JAVA_HOME}/bin:${PATH}
export M2_HOME=/opt/apache-maven-3.0.3
export PATH=${M2_HOME}/bin:${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

```
[suncoma@wso2 ~]$ echo $JAVA_HOME
/usr/java/jdk1.6.0_25
[suncoma@wso2 ~]$
```

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](#).

When using SUSE Linux, it ignores `/etc/resolv.conf` and only looks at the `/etc/hosts` file. This means that the server will throw an exception on startup if you have not specified anything besides localhost. To avoid this error, add the following line above `127.0.0.1 localhost` in the `/etc/hosts` file :

```
<ip_address> <machine_name> localhost
```

You are now ready to [run the product](#).

Installing on Windows

Follow the instructions below to install DAS on Windows.

Installing the required applications

Installation Prerequisites Make sure your system meets the .

Java Development Kit (JDK) is essential to run the product.

Installing the DAS

Download the Product [Download](#) the latest version of the DAS as described in .

2. Extract the archive file to a dedicated directory for the DAS, which will hereafter be referred to as `<DAS_HOME>`.

Installing and setting up snappy-java

1. Download the `snappy-java_1.1.1.7.jar` from [here](#).
2. Copy the jar to `<DAS_HOME>\repository\components\lib`.
3. If the DAS server is currently running, restart it to apply the changes.

Setting up JAVA_HOME

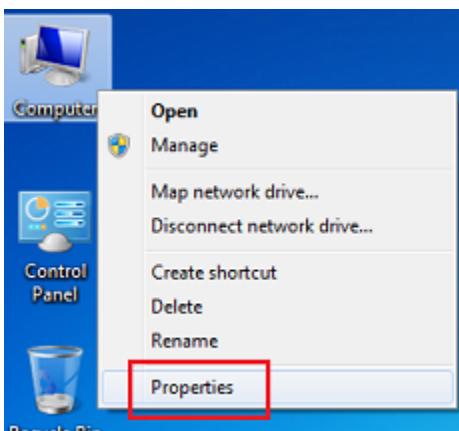
You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer. Typically, the JDK is installed in a directory under `C:\Program Files\Java`, such as `C:\Program Files\Java\jdk1.7.0_45`. If you have multiple versions installed, 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.

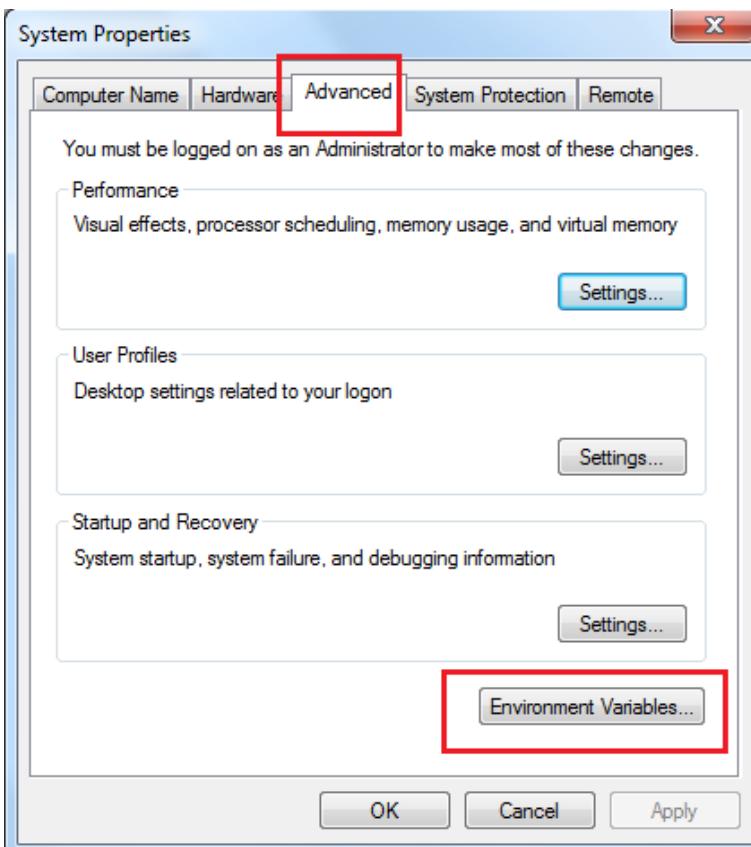
You set up `JAVA_HOME` using the System Properties, as described below. Alternatively, if you just want to set `JAVA_HOME` temporarily for the current command prompt window, [set it at the command prompt](#) .

Setting up JAVA_HOME using the system properties

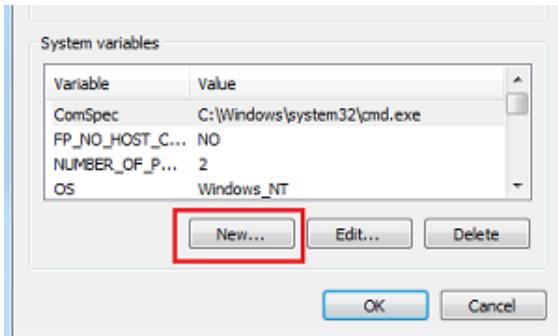
1. Right-click the **My Computer** icon on the desktop and choose **Properties**.



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\jdk1.7.0_45

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\jdk1.7.0_45`

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**.

Once all the instructions given above are completed, you may see the following error in the start up logs when you run the DAS server.

▼ [Click here to view the complete error.](#)

TID: [-1234] [] [2016-07-25 11:33:53,000] WARN {org.apache.hadoop.util.NativeCodeLoader} - Unable to load native-hadoop

library for your platform... using builtin-java classes where applicable {org.apache.hadoop.util.NativeCodeLoader}
TID: [-1234] [] [2016-07-25 11:33:53,037] ERROR {org.apache.hadoop.util.Shell} - Failed to locate the winutils binary in the
hadoop binary path {org.apache.hadoop.util.Shell}

```

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
    at org.apache.hadoop.util.Shell.getQualifiedBinPath(Shell.java:355)
    at org.apache.hadoop.util.Shell.getWinUtilsPath(Shell.java:370)
    at org.apache.hadoop.util.Shell.<clinit>(Shell.java:363)
    at org.apache.hadoop.util.StringUtils.<clinit>(StringUtils.java:79)
    at org.apache.hadoop.security.Groups.parseStaticMapping(Groups.java:104)
    at org.apache.hadoop.security.Groups.<init>(Groups.java:86)
    at org.apache.hadoop.security.Groups.<init>(Groups.java:66)
    at org.apache.hadoop.security.Groups.getUserToGroupsMappingService(Groups.java:280)
    at org.apache.hadoop.security.UserGroupInformation.initialize(UserGroupInformation.java:271)
    at org.apache.hadoop.security.UserGroupInformation.ensureInitialized(UserGroupInformation.java:248)
    at org.apache.hadoop.security.UserGroupInformation.loginUserFromSubject(UserGroupInformation.java:763)
    at org.apache.hadoop.security.UserGroupInformation.getLoginUser(UserGroupInformation.java:748)
    at org.apache.hadoop.security.UserGroupInformation.getCurrentUser(UserGroupInformation.java:621)
    at org.apache.spark.util.Utils$$anonfun$getCurrentUserName$1.apply(Utils.scala:2042)
    at org.apache.spark.util.Utils$$anonfun$getCurrentUserName$1.apply(Utils.scala:2042)
    at scala.Option.getOrElse(Option.scala:120)
    at org.apache.spark.util.Utils$.getCurrentUserName(Utils.scala:2042)
    at org.apache.spark.SecurityManager.<init>(SecurityManager.scala:212)
    at org.apache.spark.deploy.master.Master$.startSystemAndActor(Master.scala:914)
    at org.apache.spark.deploy.master.Master.startSystemAndActor(Master.scala)
    at org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor.startMaster(SparkAnalyticsExecutor.java:419)
    at
org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor.runClusteredSetupLogic(SparkAnalyticsExecutor.java:248)
    at
org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor.initializeSparkServer(SparkAnalyticsExecutor.java:174)
    at org.wso2.carbon.analytics.spark.core.internal.AnalyticsComponent.activate(AnalyticsComponent.java:71)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.eclipse.equinox.internal.ds.model.ServiceComponent.activate(ServiceComponent.java:260)
    at org.eclipse.equinox.internal.ds.model.ServiceComponentProp.activate(ServiceComponentProp.java:146)
    at org.eclipse.equinox.internal.ds.model.ServiceComponentProp.build(ServiceComponentProp.java:345)
    at org.eclipse.equinox.internal.ds.InstanceProcess.buildComponent(InstanceProcess.java:620)
    at org.eclipse.equinox.internal.ds.InstanceProcess.buildComponents(InstanceProcess.java:197)
    at org.eclipse.equinox.internal.ds.Resolver.getEligible(Resolver.java:343)
    at org.eclipse.equinox.internal.ds.SCRManager.serviceChanged(SCRManager.java:222)
    at org.eclipse.osgi.internal.serviceregistry.FilteredServiceListener.serviceChanged(FilteredServiceListener.java:107)
    at org.eclipse.framework.internal.core.BundleContextImpl.dispatchEvent(BundleContextImpl.java:861)
    at org.eclipse.osgi.eventmgr.EventManager.dispatchEvent(EventManager.java:230)
    at org.eclipse.osgi.eventmgr.ListenerQueue.dispatchEventSynchronous(ListenerQueue.java:148)
    at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEventPrivileged(ServiceRegistry.java:819)
    at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEvent(ServiceRegistry.java:771)
    at org.eclipse.osgi.internal.serviceregistry.ServiceRegistrationImpl.register(ServiceRegistrationImpl.java:130)
    at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.registerService(ServiceRegistry.java:214)
    at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:433)
    at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:451)
    at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:950)
    at
org.wso2.carbon.analytics.dataservice.core.AnalyticsDataServiceComponent.activate(AnalyticsDataServiceComponent.java:66)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.eclipse.equinox.internal.ds.model.ServiceComponent.activate(ServiceComponent.java:260)
    at org.eclipse.equinox.internal.ds.model.ServiceComponentProp.activate(ServiceComponentProp.java:146)
    at org.eclipse.equinox.internal.ds.model.ServiceComponentProp.build(ServiceComponentProp.java:345)
    at org.eclipse.equinox.internal.ds.InstanceProcess.buildComponent(InstanceProcess.java:620)
    at org.eclipse.equinox.internal.ds.InstanceProcess.buildComponents(InstanceProcess.java:197)
    at org.eclipse.equinox.internal.ds.Resolver.getEligible(Resolver.java:343)
    at org.eclipse.equinox.internal.ds.SCRManager.serviceChanged(SCRManager.java:222)
    at org.eclipse.osgi.internal.serviceregistry.FilteredServiceListener.serviceChanged(FilteredServiceListener.java:107)
    at org.eclipse.osgi.framework.internal.core.BundleContextImpl.dispatchEvent(BundleContextImpl.java:861)
    at org.eclipse.osgi.eventmgr.EventManager.dispatchEvent(EventManager.java:230)
    at org.eclipse.osgi.eventmgr.ListenerQueue.dispatchEventSynchronous(ListenerQueue.java:148)
    at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEventPrivileged(ServiceRegistry.java:819)
    at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEvent(ServiceRegistry.java:771)
    at org.eclipse.osgi.internal.serviceregistry.ServiceRegistrationImpl.register(ServiceRegistrationImpl.java:130)
```

```

at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.registerService(ServiceRegistry.java:214)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:433)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:451)
at org.wso2.carbon.ntask.core.internal.TasksDSComponent.activate(TasksDSComponent.java:106)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at org.eclipse.equinox.internal.ds.model.ServiceComponent.activate(ServiceComponent.java:260)
at org.eclipse.equinox.internal.ds.model.ServiceComponentProp.activate(ServiceComponentProp.java:146)
at org.eclipse.equinox.internal.ds.model.ServiceComponentProp.build(ServiceComponentProp.java:345)
at org.eclipse.equinox.internal.ds.InstanceProcess.buildComponent(InstanceProcess.java:620)
at org.eclipse.equinox.internal.ds.InstanceProcess.buildComponents(InstanceProcess.java:197)
at org.eclipse.equinox.internal.ds.Resolver.getEligible(Resolver.java:343)
at org.eclipse.equinox.internal.ds.SCRManager.serviceChanged(SCRManager.java:222)
at org.eclipse.osgi.internal.serviceregistry.FilteredServiceListener.serviceChanged(FilteredServiceListener.java:107)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.dispatchEvent(BundleContextImpl.java:861)
at org.eclipse.osgi.framework.eventmgr.EventManager.dispatchEvent(EventManager.java:230)
at org.eclipse.osgi.framework.eventmgr.ListenerQueue.dispatchEventSynchronous(ListenerQueue.java:148)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEventPrivileged(ServiceRegistry.java:819)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEvent(ServiceRegistry.java:771)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistrationImpl.register(ServiceRegistrationImpl.java:130)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.registerService(ServiceRegistry.java:214)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:433)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:451)
at
org.wso2.carbon.core.internal.StartupFinalizerServiceComponent.completeInitialization(StartupFinalizerServiceComponent.java:19
9)
at
org.wso2.carbon.core.internal.StartupFinalizerServiceComponent.serviceChanged(StartupFinalizerServiceComponent.java:288)
at org.eclipse.osgi.internal.serviceregistry.FilteredServiceListener.serviceChanged(FilteredServiceListener.java:107)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.dispatchEvent(BundleContextImpl.java:861)
at org.eclipse.osgi.framework.eventmgr.EventManager.dispatchEvent(EventManager.java:230)
at org.eclipse.osgi.framework.eventmgr.ListenerQueue.dispatchEventSynchronous(ListenerQueue.java:148)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEventPrivileged(ServiceRegistry.java:819)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEvent(ServiceRegistry.java:771)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistrationImpl.register(ServiceRegistrationImpl.java:130)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.registerService(ServiceRegistry.java:214)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:433)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:451)
at
org.wso2.carbon.server.admin.internal.ServerAdminServiceComponent.activate(ServerAdminServiceComponent.java:106)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at org.eclipse.equinox.internal.ds.model.ServiceComponent.activate(ServiceComponent.java:260)
at org.eclipse.equinox.internal.ds.model.ServiceComponentProp.activate(ServiceComponentProp.java:146)
at org.eclipse.equinox.internal.ds.model.ServiceComponentProp.build(ServiceComponentProp.java:345)
at org.eclipse.equinox.internal.ds.InstanceProcess.buildComponent(InstanceProcess.java:620)
at org.eclipse.equinox.internal.ds.InstanceProcess.buildComponents(InstanceProcess.java:197)
at org.eclipse.equinox.internal.ds.Resolver.getEligible(Resolver.java:343)
at org.eclipse.equinox.internal.ds.SCRManager.serviceChanged(SCRManager.java:222)
at org.eclipse.osgi.internal.serviceregistry.FilteredServiceListener.serviceChanged(FilteredServiceListener.java:107)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.dispatchEvent(BundleContextImpl.java:861)
at org.eclipse.osgi.framework.eventmgr.EventManager.dispatchEvent(EventManager.java:230)
at org.eclipse.osgi.framework.eventmgr.ListenerQueue.dispatchEventSynchronous(ListenerQueue.java:148)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEventPrivileged(ServiceRegistry.java:819)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.publishServiceEvent(ServiceRegistry.java:771)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistrationImpl.register(ServiceRegistrationImpl.java:130)
at org.eclipse.osgi.internal.serviceregistry.ServiceRegistry.registerService(ServiceRegistry.java:214)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:433)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.registerService(BundleContextImpl.java:451)
at
org.wso2.carbon.core.init.CarbonServerManager.initializeCarbon(CarbonServerManager.java:514)
at org.wso2.carbon.core.init.CarbonServerManager.removePendingItem(CarbonServerManager.java:290)
at org.wso2.carbon.core.init.PreAxis2ConfigItemListener.bundleChanged(PreAxis2ConfigItemListener.java:118)
at org.eclipse.osgi.framework.internal.core.BundleContextImpl.dispatchEvent(BundleContextImpl.java:847)
at org.eclipse.osgi.framework.eventmgr.EventManager.dispatchEvent(EventManager.java:230)
at org.eclipse.osgi.framework.eventmgr.EventManager$EventThread.run(EventManager.java:340)

```

This error does not affect the functionality of WSO2 DAS. However, you can prevent it from appearing in the logs by following the steps below.

1. Create a directory in a preferred location in your machine and name it as preferred. In this example, it is named DAS.
2. Create another directory named bin inside the directory you created.
3. Download the winutils.exe from [here](#) and place it in the DAS/bin directory you created in the previous step.
4. Specify a new Environment variable named HADOOP_HOME and set the same directory as the value (i.e. C:\DAS).

You are now ready to [run the product](#).

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.

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 ROLLOUT 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

```

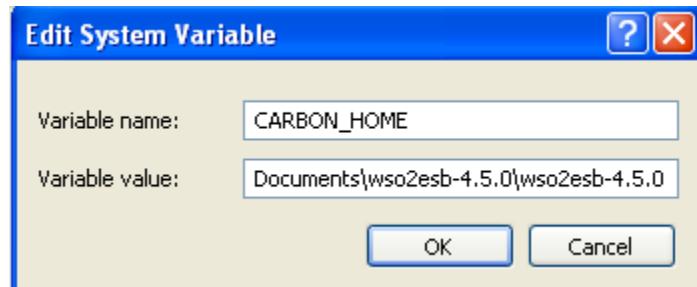
```
*****
placeHolderSoGenPropsComeHere=
wrapper.java.command = ${java_home}\bin\java
wrapper.java.classpath.1 = ${java_home}\lib\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\bin\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 = -Xbootclasspath\:/a:${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}\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 = -Dcom.sun.jndi.ldap.connect.pool.authentication=simple
wrapper.java.additional.24 = -Dcom.sun.jndi.ldap.connect.pool.timeout=3000
wrapper.java.additional.25 = -Dorg.terracotta.quartz.skipUpdateCheck=true
```

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>.

```
C:\Documents and Settings\yajsw_home\bat>runConsole.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
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" -c "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
Dec 30, 2012 11:12:27 AM org.apache.commons.vfs2.VfsLog info
INFO: Using "C:\DOCUMENTS\ADMINI\LOCALS\Temp\vfs_cache" as temporary files store.
```

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\
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
Dec 30, 2012 12:51:42 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\1\ADMINI\1\LOCALS\1\Temp\ufs_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.

```
C:\Documents and Settings\yajsw_home\bat>startService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
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" -t "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
Dec 30, 2012 1:09:00 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\1\ADMINI\1\LOCALS\1\Temp\ufs_cache" as temporary files store.
platform null
***** STARTING "WSO2CARBON" *****
Service "WSO2CARBON" started
Press any key to continue . . .

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

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.

```
C:\Documents and Settings\yajsw_home\bat>stopService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
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" -p "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
Dec 30, 2012 11:11:31 AM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
platform null
***** STOPPING "WSO2CARBON" *****
Service "WSO2CARBON" stopped
Press any key to continue . . .
```

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.

```
C:\Documents and Settings\yajsw_home\bat>uninstallService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
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" -r "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
Dec 30, 2012 1:19:14 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
platform null
***** REMOVING "WSO2CARBON" *****
Service "WSO2CARBON" removed
Press any key to continue . . .
C:\Documents and Settings\yajsw_home\bat>
```

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:

```
#!/bin/bash

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

For example, given below is a startup script written for WSO2 Application Server 5.2.0:

```
#! /bin/sh
export JAVA_HOME="/usr/lib/jvm/jdk1.7.0_07"

startcmd='/opt/WSO2/wso2as-5.2.0/bin/wso2server.sh start > /dev/null &'
restartcmd='/opt/WSO2/wso2as-5.2.0/bin/wso2server.sh restart > /dev/null &'
stopcmd='/opt/WSO2/wso2as-5.2.0/bin/wso2server.sh stop > /dev/null &

case "$1" in
start)
    echo "Starting WSO2 Application Server ..."
    su -c "${startcmd}" user1
;;
restart)
    echo "Re-starting WSO2 Application Server ..."
    su -c "${restartcmd}" user1
;;
stop)
    echo "Stopping WSO2 Application Server ..."
    su -c "${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 appserver 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/appserver`
- Add a link to `/etc/init.d/`: `sudo ln -snf /opt/WSO2/appserver /etc/init.d/appserver`

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 appserver 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.

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 do [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-das`

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:

IDE	Command	Additional information
Eclipse	<code>mvn eclipse:eclipse</code>	http://maven.apache.org/plugins/maven-eclipse-plugin
IntelliJ IDEA	<code>mvn idea:idea</code>	http://maven.apache.org/plugins/maven-idea-plugin

2. Add the required changes to the source code.

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:

Command	Description
<code>mvn clean install</code>	The binary and source distributions.
<code>mvn clean install -Dmaven.test.skip=true</code>	The binary and source distributions, without running any of the unit tests.
<code>mvn clean install -Dmaven.test.skip=true -o</code>	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.

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 DAS 3.0.1

This section explains how to upgrade to DAS 3.1.0 from DAS 3.0.1. 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 Analytics tables
- Migrating the configurations
- Migrating index information
- Migrating artifacts
- Testing the upgrade

Preparing to upgrade

The following prerequisites should be completed before upgrading.

- Make a backup of the DAS 3.0.1 database and copy the <DAS_HOME_3.0.1> directory in order to backup the product configurations.
- Download any CAR files that are deployed in the DAS server and save them. Then delete them from DAS 3.0.1. This removes the scheduled tasks relating to the deployed artifacts and allows migration to be carried out smoothly. For more information, see [Packaging Artifacts as a C-App Archive](#).
- Download WSO2 DAS 3.1.0 from <http://wso2.com/products/data-analytics-server/>.
- Make the required changes in the datasource files in the <DAS_HOME>/repository/conf/datasources directory to make sure that they are configured as required. For more information, see [Datasources](#).

Migrating Analytics tables

The table schema internally used by DAS 3.1.0 is different to that of DAS 3.0.x. Therefore, this meta data transformation should be done before using the Analytics tables in DAS 3.0.x in DAS 3.1.0 using the following procedure. The stored data itself is backward compatible. Therefore, it does not go through all the data. Instead, a constant time metadata conversion in tables is carried out.

1. Navigate to the <DAS_HOME>/bin directory.
2. Issue one of the following commands as appropriate.
 - On Windows: analytics-backup.bat -migrateTableSchemaV30To31
 - On Linux/Solaris/Mac OS: ./analytics-backup.sh -migrateTableSchemaV30To31

Migrating the configurations

Once you have completed the database upgrade, You can proceed with the configuration changes as explained below. The configurations can be directly migrated from DAS 3.0.1 to DAS 3.1.0.

Configurations should not be copied directly between servers.

To connect DAS 3.1.0 to the upgraded database, configure the following files:

1. Go to the <DAS_HOME>/repository/conf directory and update the datasource references in the `user-mgt.xml` and `registry.xml` files to match the updated configurations in the `master-datasources.xml` file that you made in the above step.
For detailed instructions to configure the `user-mgt.xml` file, see [User Management](#).
For detailed instructions to configure the `registry.xml` file, see [Registry](#).
2. Compare the configuration files in the < DAS_HOME>/repository/conf/data-bridge directory for DAS 3.0.1, and 3.1.0.
3. Compare the configuration files in the < DAS_HOME>/repository/conf/analytics directory for DAS 3.0.1, and 3.1.0. Change the default values in DAS 3.1.0 to the same values you have specified in DAS 3.0.1.
4. Change the default values in <DAS_3.1.0_HOME>/repository/conf/event-processor.xml file to the same values you have specified in the <DAS_3.0.1_HOME>/repository/conf/event-processor.xml file.

If you enable the HA mode for WSO2 DAS by setting the `<mode name="HA" enable="true">` property in the `event-processor.xml` file, state persistence is enabled by default. If there is no real time use case that requires any state information after starting the cluster, you should disable event persistence by setting the `persistence` attribute to `false` in the same file as shown below.

```

<persistence enable="false">
    <persistenceIntervalInMinutes>15</persistenceIntervalInMinutes>
    <persisterSchedulerPoolSize>10</persisterSchedulerPoolSize>
    <persister
        class="org.wso2.carbon.event.processor.core.internal.persistence.FileSystemPersistenceStore">
        <property key="persistenceLocation">cep_persistence</property>
    </persister>
</persistence>

```

- Check for any other configurations that were done for DAS 3.0.1 based on your solution and update the configuration files in DAS 3.1.0 accordingly. For example, configurations related to external user stores, caching, mounting, transports etc.

Migrating index information

Each node stores its index data in the <DAS_HOME>/repository/data directory. There are two possible approaches to migrate this data. Click on the relevant tab based on the approach you want to follow.

- Using re-indexing command
- Copying index data

This method involves running the analytics data backup/restore tool with the `-reindexEvents` setting to index data. Here, the indexing is done using the existing data in the database to which you already connected WSO2 DAS 3.1.0 in the [Migrating the configurations](#) section. For more information, see [Analytics Data Backup / Restore Tool](#). Follow the steps below to migrate the index information from DAS 3.0.1 to DAS 3.1.0.

- Copy the contents of the <DAS_3.0.1>/repository/data directory and replace the contents of the <DAS_3.1.0>/repository/data directory with it.
- Take copies of the following files in the <DAS_3.0.1_HOME> and use them to replace the same files located in the <DAS_3.1.0_HOME>
 - <DAS_HOME>/repository/conf/analytics/my-node-id.dat
 - <DAS_301>/repository/conf/analytics/local-shard-allocation-config.conf

You should not mix the configuration files between nodes. The configuration files and index data of DAS 3.0.1 - node 1 should be copied to DAS 3.1.0 - node 1, and the configuration files and index data of DAS 3.0.1 - node 2 should be copied to DAS 3.1.0 - node 2 etc.

Migrating artifacts

All artifacts can be migrated from DAS 3.0.1 to DAS 3.1.0 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 DAS 3.0.1 to DAS 3.1.0.

To migrate deployment artifacts:

- Copy the <DAS_HOME>/repository/deployment/server directory from DAS 3.0.1 to DAS 3.1.0.

- If multi-tenancy is used, copy the <DAS_HOME>/repository/tenants directory from DAS 3.0.1 to DAS 3.1.0.

Additional steps are required to migrate Spark scripts and dashboard elements and explained in the following subsections.

Updating Spark scripts

The CarbonJDBC provider was rewritten for DAS 3.1.0. Therefore, the following additional input parameters are required to be added to the Spark Scripts migrated to DAS 3.1.0 for which Carbon JDBC is used as the provider.

- **schema** (Required): The schema to be used within Spark for the specifies table.
- **primaryKeys** (optional): Any database level unique keys that are either already specified or needed for the table.

The following example shows the difference in the query format between DAS 3.0.1 and DAS 3.1.0.

DAS 3.0.1

```
CREATE TEMPORARY TABLE <temp_table> using CarbonJDBC options (dataSource "<datasource name>" , tableName "<table name>" );
```

DAS 3.1.0

```
CREATE TEMPORARY TABLE <temp_table> using CarbonJDBC options (dataSource "<datasource name>" , tableName "<table name>" , schema "<schema>" [, primaryKeys "<primaryKeys>" ]);
```

For more information about improvements made to the CarbonJDBC provider, see [Creating the Table Using Carbon JDBC as the Provider](#).

Updating execution plans

WSO2 DAS 3.1.0 uses Siddhi 3.1.2 whereas DAS 3.0.1 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 DAS 3.0.1 to DAS 3.1.0, 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 DAS 3.0.1.

DAS 3.0.1

```
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 DAS 3.1.0, #window.length(1) property should be manually added to it as shown below.

DAS 3.1.0

```
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

The Analytics Dashboard functionality in DAS 3.1.0 is powered by WSO2 Dashboard Server 2.1.0 whereas the same functionality in DAS 3.0.1 and earlier versions is powered by WSO2 Gadget Server 1.4.0. This section explains how to migrate the gadgets and dashboards from DAS 3.0.1 to 4.2.0.

There are two methods that can be used to migrate gadgets and dashboards between the two DAS 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.

File Name	Path
ds-migration-1.0.x-2.0.x.bat or ds-migration-1.0.x-2.0.x.sh	<DAS_HOME>/migration/dashboards/migration-1.0.x-2.0.x
migration.xml	<DAS_HOME>/migration/dashboards/migration-1.0.x-2.0.x

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.

1. Open the `<DAS_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.

```

<!-- 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 direcotory.-->
        <!-- <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.jk
s</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 -->
        <!-- <DestinationURL>https://localhost:9444</DestinationURL> -->

        <!-- 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 anything here, It will
update all the tenant dashboards -->
        <!-- <TenantDomains></TenantDomains>-->
    <!-- </DSMigration> -->

```

2. Run the following command to run the migration script.

For Windows: <DAS-3.0.1_HOME>/migration/dashboards/migration-1.0.x-2.0.x/ds-migration-1.0.x-2.0.x.bat --run

For Linux: <DAS-3.0.1_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 DAS 3.0.1 to a CAR file and deploy it in WSO2 DAS 3.1.0. For more information, see [Packaging Artifacts as C-App Archive](#).
4. Perform the following steps for the real time gadgets (i.e. gadgets that use real time event streams as their information source).

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

- a. Navigate to the home directory of the gadget. (e.g., <DESTINATION_DIRECTORY>/store/carbon.super/fs/gadget/my_gadget).
- b. Replace the <DESTINATION_DIRECTORY>/store/carbon.super/fs/gadget/my_gadget/js/main.js file with the <DAS_HOME>/migration/dashboards/migration-1.0.x-2.0.x/resources/main.js file.
- c. Similarly, replace the <DESTINATION_DIRECTORY>/store/carbon.super/fs/gadget/my_gadget/js/outputAdapterUiLibrary.js file with <DAS_HOME>/migration/dashboards/migration-1.0.x-2.0.x/resources/outputAdapterUiLibrary.js file.

In this approach the product home directory is defined as the source file. It migrates gadgets, layouts and widgets into the newer DAS 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 <DAS_HOME>/migration/dashboards/migration-1.0.x-2.0.x/migration.xml file, and comment out the CAR File Migration section as shown in the example below.

```

<!-- CAR File Migration-->
<!-- Uncomment the below part to enable CAR file migration -->
<DSMigration>
    <!-- Mode - Specifiy the migration type CAR or Portal -->
    <Mode>CAR</Mode>
    <!-- SourceDir - Specify the older PRODUCT_HOME direcotory.-->
    <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.

Parameter	Description	Example
-----------	-------------	---------

Mode	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).	
SourceDir	The source of the dashboard elements to be migrated. The < DAS_3.0.1_HOME> should be specified as the source directory.	
DestinationDir	The destination directory for the migrated Store.	
TrustStoreLocation	This parameter specifies the trusted store location.	<DAS_HOME>/repository/resources/security/v
TrustStorePassword	The password of the trust store.ashboard	wso2carbon
SourceURL	The source URL. This should be the URL of the DAS 3.0.1 instance from which you are migrating the dashboard elements.	https://hostname:port
SourceUsername	Specify the username of the carbon super user.	admin
SourcePassword	Specify the password of the carbon super user.	admin

DestinationURL	The destination URL. This should be the URL of the DAS 3.1.0 to which you are migrating the dashboard elements.	https://hostname:port
DestinationUsername	Specify the username of the carbon super user.	admin
DestinationPassword	Specify the password of the carbon super user.	admin
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.

Option	Description	Required Parameters	Configuration

1	<p>Convert gadgets, widgets, and layouts in DAS 3.0.1 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 DAS dashboards store.</p> <ul style="list-style-type: none"> • Mode • SourceDir • DestinationDir 		<pre><DSMmigration> <Type>Portal</Type> <SourceDir>YOUR_PRODUCT_HOME</SourceDir> <DestinationDir>YOUR_DESTINATION_DIR</DestinationDir> <TrustStoreLocation></TrustStoreLocation> <TrustStorePassword></TrustStorePassword> <SourceURL></SourceURL> <SourceUsername></SourceUsername> <SourcePassword></SourcePassword> <DestinationURL></DestinationURL> <DestinationUsername></DestinationUsername> <DestinationPassword></DestinationPassword> <TenantDomains></TenantDomains> </DSMmigration></pre>
---	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2	<p>Convert gadgets, widgets, and layouts in DAS 3.0.1 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.</p>	<ul style="list-style-type: none"> • Mode • SourceDir • DestinationDir • TrustStoreLocation • TrustStorePassword • SourceURL • SourceUsername • SourcePassword 	<pre> <DSMmigration> <Type>Portal</Type> <SourceDir>YOUR_PRODUCT_HOME</SourceDir> <DestinationDir>YOUR_DESTINATION_HOME</DestinationDir> <TrustStoreLocation>YOUR_PRODUCT_HOME</TrustStoreLocation> <TrustStorePassword>wso2carbon</TrustStorePassword> <SourceURL>https://localhost:9443/dashboards</SourceURL> <SourceUsername>admin</SourceUsername> <SourcePassword>admin</SourcePassword> <DestinationURL></DestinationURL> <DestinationUsername></DestinationUsername> <DestinationPassword></DestinationPassword> <TenantDomains></TenantDomains> </DSMmigration> </pre>
---	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3	<p>Convert gadgets, widgets, and layouts in DAS 3.0.1 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.</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p>This requires you to maintain the same trust store location and the trust password in both DAS versions.</p> </div>	<ul style="list-style-type: none"> • Mode • SourceDir • DestinationDir • TrustStoreLocation • TrustStorePassword • SourceURL • SourceUsername • SourcePassword • DestinationURL • DestinationUsername • DestinationPassword 	<pre><DSMmigration> <Type>Portal</Type> <SourceDir>YOUR_PRODUCT_HOME</SourceDir> <DestinationDir>YOUR_DESTINATION_DIR</DestinationDir> <TrustStoreLocation>YOUR_PRODUCT_HOME</TrustStoreLocation> <TrustStorePassword>wso2carbon</TrustStorePassword> <SourceURL>https://localhost:9443/dashboards</SourceURL> <SourceUsername>admin</SourceUsername> <SourcePassword>admin</SourcePassword> <DestinationURL>https://localhost:9443/dashboards</DestinationURL> <DestinationUsername>admin</DestinationUsername> <DestinationPassword>admin</DestinationPassword> <TenantDomains></TenantDomains> </DSMmigration></pre>
---	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3. Run the following command to run the migration script.

For Windows: <DAS-3.0.1_HOME>/migration/dashboards/migration-1.0.x-2.0.x/ds-migration-1.0.x-2.0.x.bat --run

For Linux: <DAS-3.0.1_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 <DAS_HOME>/migration/dashboard/migration-1.0x-2.0.x/migration.xml file.

- a. Navigate to the home directory of the gadget. (e.g., <DESTINATION_DIRECTORY>/store/carbon).

- super/fs/gadget/my_gadget).
- Replace the <DESTINATION_DIRECTORY>/store/carbon.super/fs/gadget/my_gadget/js/main.js file with the <DAS_HOME>/migration/dashboards/migration-1.0.x-2.0.x/resources/main.js file.
 - Similarly, replace the <DESTINATION_DIRECTORY>/store/carbon.super/fs/gadget/my_gadget/js/outputAdapterUiLibrary.js file with <DAS_HOME>/migration/dashboards/migration-1.0.x-2.0.x/resources/outputAdapterUiLibrary.js file.

Validating path changes

DAS 3.1.0 has a different directory structure to DAS 3.0.1 that affects the path to the gadget store as shown below.

DAS 3.0.1: <DAS_HOME>/repository/deployment/server/jaggeryapps/portal/store/<Tenant_Name> / g a d g e t

DAS 3.1.0: <DAS_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 <DAS_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 DAS 3.0.1.

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

The same link should be given as follows in DAS 3.1.0 due to the change in the directory structure (i.e. an additional .. should be added for the new directory added to the structure).

```
<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 DAS version.

DAS Version	3.0.1
Location	<DAS_HOME>/repository/deployment/server/jaggeryapps/portal/store/<Tenant_Name>
Indexing	1 based indexing

Format

```
{
  "blocks": [
    { "id": "a", "row": 1, "col": 1, "size_x": 4, "size_y": 3 },
    { "id": "b", "row": 1, "col": 5, "size_x": 4, "size_y": 3 },
    { "id": "c", "row": 1, "col": 9, "size_x": 4, "size_y": 3 },
    { "id": "d", "row": 4, "col": 1, "size_x": 4, "size_y": 3 },
    { "id": "e", "row": 4, "col": 5, "size_x": 4, "size_y": 3 },
    { "id": "f", "row": 4, "col": 9, "size_x": 4, "size_y": 3 }
  ]
}
```

The layouts migrated from DAS 3.0.1 to DAS 3.1.0 will have the format in which layouts are saved in DAS 3.0.1. To make them compatible with DAS 3.1.0

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 <PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml file. (<PRODUCT_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

Follow the instructions below to start your WSO2 product based on the Operating System you use.

On Windows/Linux/Mac OS

To start the server, you run <PRODUCT_HOME>/bin/wso2server.bat (on Windows) or <PRODUCT_HOME>/bin/wso2server.sh (on Linux/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/Mac OS: Establish an SSH connection to the server, log on to the text Linux console, or open a terminal window.
2. Navigate to the `<PRODUCT_HOME>/bin/` directory using the Command Prompt.
 3. Execute one of the following commands:
 - To start the server in a typical environment:
 - On Windows: `wso2server.bat --run`
 - On Linux/Mac OS: `sh wso2server.sh`
 - To start the server in the background mode of Linux: `sh wso2server.sh start`
To stop the server running in this mode, you will enter: `sh wso2server.sh stop`
 - To provide access to the production environment without allowing any user group (including admin) to log in to the Management Console:
 - On Windows: `wso2server.bat --run -DworkerNode`
 - On Linux/Mac OS: `sh wso2server.sh -DworkerNode`
 - To check for additional options you can use with the startup commands, type `-help` after the command, such as:
`sh wso2server.sh -help` (see the related topics section at the end of this page).

4. 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".

On Solaris

To start the server, you run `<PRODUCT_HOME>/bin/wso2server.sh` from the Command Prompt as described below.

Following instructions are tested on an Oracle Solaris 10 8/11 x86 environment.

1. Click **Launch -> Run Applications**, type `dtterm` at the Prompt, and then press **Enter**, to open a Command Prompt.
2. Navigate to the `<PRODUCT_HOME>/bin/` directory using the Command Prompt.
3. Execute the following command: `bash wso2server.sh`
4. 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 starting the product in service/nohup mode in Solaris, do the following:

1. Update the `<PRODUCT_HOME>/bin/wso2server.sh` file as follows:
 - a. Search for the following occurrences: `nohup sh "$CARBON_HOME"/bin/wso2server.sh $args > /dev/null 2>&1 &`
 - b. Replace those occurrences with the following: `nohup bash "$CARBON_HOME"/bin/wso2server.sh $args > /dev/null 2>&1 &`

The only change is replacing `sh` with `bash`. This is required only for Solaris.

2. Update your **PATH** variable to have `/usr/xpg4/bin/sh` as the first element. This is because `/usr/xpg4/bin/sh` contains an `sh` shell that is newer than the default `sh` shell. You can set this variable as a system property in the `wso2server.sh` script or you can run the following command on a terminal:

```
export PATH=/usr/xpg4/bin/sh:$PATH
```

3. Start the product by following the above instructions.

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 17:53:26,547] INFO {org.wso2.carbon.core.internal.StartupFinalizerServiceComponent} - WSO2 Carbon started in 45 sec
[2014-12-04 17:53:26,787] INFO {org.wso2.carbon.ui.internal.CarbonUIServiceComponent} - Mgt Console URL : https://localhost:9443/carbon/
```

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 `<PRODUCT_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 <PRODUCT_HOME>/repository/conf/tomcat/carbon/WEB-INF/web.xml file as follows.

```
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
```

In products like WSO2 API Manager where web applications such as API Publisher/API Store exist, you can configure a session time out for those web apps by changing the repository/conf/tomcat/web.xml file as follows:

```
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
```

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 <PRODUCT_HOME>/repository/conf/tomcat/carbon/META-INF/context.xml file as follows:

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
allow="|<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 <PRODUCT_HOME>/repository/conf/context.xml file as follows.

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
allow="|<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 <PRODUCT_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.

```
<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>127.0.0.1</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 values defined in the `<PRODUCT_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/Restart** 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 <PRODUCT_HOME>/bin/wso2server.sh stop
```

WSO2 DAS Production Setup Checklist

The following is a checklist you can follow before using WSO2 DAS in a production environment.

- **Capacity Planning**

This involves determining the scale at which you need to deploy DAS based on your requirement. To see the output of WSO2 DAS at different scales, see [WSO2 DAS Performance Analysis](#).

- **Clustering**

This involves selecting the clustering pattern which matches your requirements and setting up the cluster accordingly. For more information on deployment options available for clustering WSO2 DAS, see [Clustering Data Analytics Server 3.0.0](#).

- **Performance Tuning**

This involves configuring the relevant parameters to optimize the performance of WSO2 DAS depending on your production environment. For detailed information about the parameters to be configured, see the [Performance Tuning](#).

- **Providing Access**

This involves providing the required level of access to all the users in your production environment. For more information, see [User Management](#).

- **Securing**

This involves securing your production environment as described in [Security](#).

- **Registry Configuration**

This involves identifying your registry requirements and configuring registries as required. For more information, see [Registry](#).

- **Installing Features**

This involves checking whether all the features you require are already available by default and installing any features that are missing. For more information, see [Feature Management](#).

User Guide

The main functionality of WSO2 DAS can be divided into 3 parts as data **aggregating**, **analyzing**, and **presenting** the analyzed data through various dashboards and gadgets. Take a look at how these components interact and how the message flow happens in section Architecture.

The user guide provides information about the features, functionality, solution development, testing and debugging options of WSO2 DAS.

- [Understanding Event Streams and Event Tables](#)
- [Collecting Data](#)
- [Analyzing Data](#)
- [Communicating Results](#)
- [Packaging Artifacts as a C-App Archive](#)
- [Debugging](#)
- [Managing DAS Artifacts via Template Manager](#)

[Go to Home Page](#)

Understanding Event Streams and Event Tables

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 DAS.

- [Event streams](#)
- [Event formats](#)
- [Event Flow](#)
- [Analytics event table](#)

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.

```
{
  "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.

Property	Description
Event Stream Name	Name of the event stream.
Event Stream Version	Version of the event stream. (Default value is 1.0.0.)
Event Stream Description	Description of the events stream. (This is optional.)
Event Stream Nick-Name	Nick-names of an event streams separated by commas.(This is optional.)

Stream Attributes	<p>Stream attributes contains the data of the event. Data is divided into the following 3 logical categories for maintenance and usability. It is not required to have attributes for all 3 categories, but there should be at least one category with at least one attribute defined. The attribute names should be unique within each category.</p> <ul style="list-style-type: none"> • Meta Data: This refers to meta information of the events (e.g., timestamp, host, IP, etc.). They are received with the <code>meta_<attribute name></code> format. • Correlation Data: This refers to information that is used to correlate multiple events (e.g., <code>correlation_id</code>, <code>temporal_id</code> etc.). They are received with the <code>correlation_<attribute name></code> format. • Payload Data: Contains the actual data that the event intends to have. (Referred to as <code><attribute name></code>.) <p>e.g., The following attributes exist in a single event.</p> <ul style="list-style-type: none"> • <code>event_timestamp</code> • <code>request_IP_address</code> • <code>correlation_Id</code> • <code>price</code> • <code>symbol</code> <p>These attributes can be logically categorized as follows.</p> <table border="1"> <thead> <tr> <th>Category</th><th>Attributes</th><th>Notes</th></tr> </thead> <tbody> <tr> <td>Meta Data</td><td> <ul style="list-style-type: none"> • <code>event_timestamp</code> • <code>request_IP_address</code> </td><td>These attributes provide information about the events themselves.</td></tr> <tr> <td>Correlation Data</td><td> <ul style="list-style-type: none"> • <code>correlation_Id</code> </td><td>This attribute correlates the event with other events from other streams, and it is useful when performing join operations on a stream.</td></tr> <tr> <td>Payload Data</td><td> <ul style="list-style-type: none"> • <code>price</code> • <code>symbol</code> </td><td>These are information provided via the event.</td></tr> </tbody> </table> <p> <ul style="list-style-type: none"> • Note that the internal system does not differentiate the attributes based on this categorization. However, it is recommended to categorize the attributes for the purpose of organizing them in a logical manner within the stream definition. • If you edit an attribute in an event stream or add a new attribute to it, other artifacts that are associated with the stream will be inactive. Therefore, it is recommended to redefine the stream with a new version including the additional/changed attribute. </p>	Category	Attributes	Notes	Meta Data	<ul style="list-style-type: none"> • <code>event_timestamp</code> • <code>request_IP_address</code> 	These attributes provide information about the events themselves.	Correlation Data	<ul style="list-style-type: none"> • <code>correlation_Id</code> 	This attribute correlates the event with other events from other streams, and it is useful when performing join operations on a stream.	Payload Data	<ul style="list-style-type: none"> • <code>price</code> • <code>symbol</code> 	These are information provided via the event.
Category	Attributes	Notes											
Meta Data	<ul style="list-style-type: none"> • <code>event_timestamp</code> • <code>request_IP_address</code> 	These attributes provide information about the events themselves.											
Correlation Data	<ul style="list-style-type: none"> • <code>correlation_Id</code> 	This attribute correlates the event with other events from other streams, and it is useful when performing join operations on a stream.											
Payload Data	<ul style="list-style-type: none"> • <code>price</code> • <code>symbol</code> 	These are information provided via 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.

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	org.wso2.test <small>Name of the Event Stream</small>
Event Stream Version*	1.0.0 <small>Version of the event stream (Eg : 1.0.0)</small>
Event Stream Description	Test Stream <small>Description of the event stream</small>
Event Stream Nick-Name	TestStream <small>Nick of the event stream</small>

Stream Attributes

Meta Data Attributes

Attribute Name	Attribute Type	Actions
ip	string	Delete

Attribute Name : Attribute Type :

Correlation Data Attributes

Attribute Name	Attribute Type	Actions
id	long	Delete

Attribute Name : Attribute Type :

Payload Data Attributes

Attribute Name	Attribute Type	Actions
testMessage	string	Delete

Attribute Name : Attribute Type :

[Add Event Stream](#)

- 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.

[Home](#) > [Manage](#) > [Event Processor](#) > [Event Streams](#)

Available Event Streams

[Add Event Stream](#)

2 Event streams available

Event Stream Id	Event Stream Description	Actions
org.wso2.test:1.0.0	Test Stream	
test:1.0.0		

Using the source view

Follow the steps below to add an event stream using the source view.

- Log in to the management console, and click **Main**.
- Click **Event Streams** in the **Event Processor** menu, and then click **Add Event Stream**.
- Click **switch to source view**.

Click **switch to design view** to add the event stream using the design view.

- Enter details of the stream definition that you want to create as shown in the below example.

[Home](#) > [Manage](#) > [Event Processor](#) > [Event Streams](#)

Define New Event Stream

Enter Event Stream Details [switch to design view](#)

```
{
  "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"
    }
  ]
}
```

[Add Event Stream](#)

- Click **Add Event Stream**, to create the event stream in the system. You can view the new event stream in the **Available Event Streams** list as shown below.

[Home](#) > [Manage](#) > [Event Processor](#) > [Event Streams](#)

Available Event Streams

[Add Event Stream](#)

2 Event streams available

Event Stream Id	Event Stream Description	Actions
org.wso2.test:1.0.0	Test Stream	Delete Edit
test:1.0.0		Delete Edit

Deleting an event stream

Follow the steps below to delete an event stream by deleting the corresponding event stream definition.

- Log in to the management console, and click **Main**.
- Click **Event Streams** in the **Event Processor** menu. You view the **Available Event Streams** list.
- 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.

```
{
  "event": {
    "metaData": {
      "ip": "data3"
    },
    "correlationData": {
      "id": "56783"
    },
    "payloadData": {
      "testMessage": "data4"
    }
  }
}
```

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

```
<events>
  <event>
    <metaData>
      <ip>data4</ip>
    </metaData>
    <correlationData>
      <id>56783</id>
    </correlationData>
    <payloadData>
      <testMessage>data1</testMessage>
    </payloadData>
  </event>
</events>
```

Default JSON format

```
{
    "event": {
        "metaData": {
            "ip": "data4"
        },
        "correlationData": {
            "id": "545455"
        },
        "payloadData": {
            "testMessage": "data1"
        }
    }
}
```

Default text format

```
meta_ip:data1,
correlation_id:323232,
testMessage:data2
```

Default map format

Key	Value
meta_ip	data1
correlation_id	323232
testMessage	data2

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

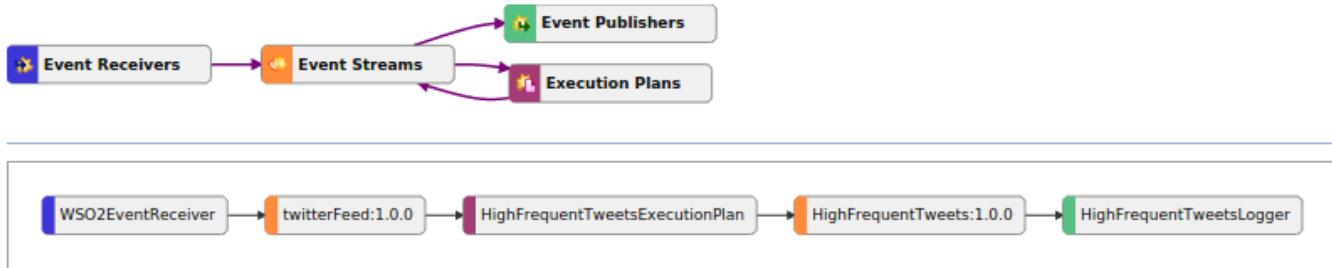
Event flow visualizes the stream flow in WSO2 CEP/DAS to easily navigate to different WSO2 CEP/DAS components.

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.

CEP Event Flow



Analytics event table

In batch analytics, an event table is used to persist events from a stream, and to later lookup/update/delete from it. The Data Analytics Server contains an event table implementation based on its Data Access Layer, where users can create an event table based on the underlying configured data source of the server.

The analytics event table will follow the best approach in carrying out its tasks, e.g. using the primary keys if there are no non-equal conditional expressions in the queries etc.. or else, for conditions that require less than, greater than, less than or equal, greater than or equal, contains, then the respective fields must be marked as indexed.

The syntax in using the analytics event table is as follows:-

```

@from(eventtable = 'analytics.table' , table.name = <analytics_table_name>,
primary.keys = <primary_keys>, indices = <indices>, wait.for.indexing =
<wait_for_indexing_flag>, merge.schema = <merge_schema_flag>)
define table <EventTableName> (<schema>);

```

Field Name	Description	Required	Default Value
table.name	The name of the analytics table, this can be an existing table, or else, a new one will be created.	Yes	
primary.keys	The list of fields to be used as the primary keys of the table, this can be useful, if the lookup operations are done only using primary key values, which is the most efficient to execute.	No	
indices	The list of index fields separated by commas, each entry consists of the format "<index_column_name> -sp", where "-sp" is optional property to say, if this index column should be treated as a score param.	No	

wait.for.indexing	The indexing operations in the analytics tables happens asynchronously, if events coming from a specific stream changing the event table's data needs to be finalized, setting this flag to 'true' would wait till the background indexing to finish and continue with the execution of the flow.	No	true
merge.schema	In the case of an existing table given to the analytics event table, if this flag is set to 'true', the existing schema and the given schema will be merged together. That is, the merging of columns and its indexing information, if set to 'false', the schema given by the analytics event table will overwrite the existing one.	No	true
caching	Enables caching for the analytics table. This will store any looked up data from the analytics table and store the most recently accessed data, with the given capacity and timeout restrictions of the cache.	No	false
cache.timeout.seconds	The timeout of the cache entries in seconds	No	10
cache.size.bytes	The maximum capacity of the cache in bytes	No	10485760

Sample

```
@from(eventtable = 'analytics.table' , table.name = 'stocks', primary.keys = 'symbol',
indices = 'price, volume -sp', wait.for.indexing = 'true', merge.schema = 'false')
define table StockTable (symbol string, price float, volume long);
```

Configuring Data Persistence

WSO2 DAS introduces the ability to have a pluggable Data Access Layer (DAL).

Analytics Record Store

The Analytics Record Store is the section that handles the storing of records that are received by WSO2 DAS in the form of events. This store contains raw data relating to events in a tabular form to be retrieved later.

The following record stores are configured in the <DAS_HOME>/repository/conf/analytics/analytics-config.xml file by default.

Record Store Type	Default Name	Description
Primary Store	EVENT_STORE	This record store is used to store the persisted incoming events of WSO2 DAS. It contains raw data in a tabular structure which can be used later.
Processed Data Store	PROCESSED_DATA_STORE	This record store is used to store summarized event data.

Configuring a record store

The following is a sample configuration of a record store.

```
<analytics-record-store name="EVENT_STORE">

<implementation>org.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsRecordStore</
implementation>
<properties>
    <property name="datasource">WSO2_ANALYTICS_EVENT_STORE_DB</property>
    <property name="category">large_dataset_optimized</property>
</properties>
</analytics-record-store>
```

The following needs to be specified for each record store.

- **Name:** A unique name for the record store.
- **Implementation:** This specifies the implementation for the record store, this class should be implementing the interface "org.wso2.carbon.analytics.datasource.core.rs.AnalyticsRecordStore". For the record store to function, the provider for the datasource type mentioned in this implementation should be enabled in the `<DA S_HOME>/repository/conf/datasources/analytics-datasources.xml` file.
- **Record store specific properties:** The properties that are defined per record store are described in the table below.

Property	Description
datasource	The name of the datasource used to connect to the database used by the record store.

Once a record store is configured in the `analytics-config.xml` file, you can select it as the record store for the required event streams. For more information, see [Persisting Data for Interactive Analytics](#).

Analytics Indexing

By default, WSO2 DAS executes indexing operation when the server is started. The following system property can be used to disable the indexing operations if required.

- For Windows: `wso2server.bat -DdisableIndexing`
- For Linux: `wso2server.sh -DdisableIndexing`

This option allows you to create servers that are dedicated for specific operations such as event receiving, analytics, indexing, etc.

Configuring common parameters

Data purging parameters

Parameter	Description	Default Value
<code><purging-enable></code>	This parameter specifies whether the functionality to purge data from event tables is enabled or not.	false
<code><cron-expression></code>	A regex expression to select the tables from which data should be purged.	0 0 0 * * ?
<code><purge-include-tables></code>	A list of event tables from which data should be purged can be defined as subelements of this element.	

<data-retention-days>	The number of days for which the data should be retained in the event tables that were selected to have their data purged. All the data in these tables are cleared once the number of days that equal the value specified for this parameter have elapsed.	365
-----------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----

Other Parameters

Parameter	Description
<analytics-lucene-analyzer>	The implementation of the Analytics Lucene Analyzer is defined as a sub parameter. e.g., <implementation>org.apache.lucene.analysis.standard.StandardAnalyzer
<indexReplicationFactor>	The index data replication factor to be used in clustered mode. This tells how that should be kept when indexing operations are done. 0 means, no replication should be set to 1 or higher to have high availability.
<shardCount>	The number of index shards the server should maintain per cluster. This defines the scaling nature of the indexing cluster. This parameter can only be set once for the lifetime of the cluster, and cannot be changed later on.
<shardIndexRecordBatchSize>	The amount of index data (in bytes) to be processed at a time by a shard index processing worker. Minimum value is 1000.
<shardIndexWorkerInterval>	The interval in milliseconds during which a shard index processing worker takes up work during index processing operations. This parameter together with the shardIndexRecordBatchSize parameter can be used to increase the final index batch size processed by the indexer at a given time. Usually, a higher batch data amount leads to a higher throughput of the indexing operations. However, it also causes an increase in the number of record insertion to indexing. Minimum value of this is 10, and a maximum value of 1000 (1 minute).
<indexWorkerCount>	The number of index workers to operate in the current node. This basically defines the number of execution threads created to do the indexing operations of the system. When this value is increased, the parallel I/O operations being done on the system will be larger. So a system which can handle parallel I/O operation could increase the throughput.

The DAL configuration can be found at <DAS_HOME>/repository/conf/analytics/analytics-config.xml. An example is shown below.

sample analytics-config.xml

```

<analytics-dataservice-configuration>
    <!-- The name of the primary record store -->
    <primaryRecordStore>EVENT_STORE</primaryRecordStore>
    <!-- Analytics Record Store - properties related to record storage implementation
-->

```

```

<analytics-record-store name="EVENT_STORE">

<implementation>org.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsRecordStore</
implementation>
    <properties>
        <property name="datasource">WSO2_ANALYTICS_EVENT_DB</property>
        <property name="category">large_dataset_optimized</property>
    </properties>
</analytics-record-store>
<analytics-record-store name = "PROCESSED_DATA_STORE">

<implementation>org.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsRecordStore</
implementation>
    <properties>
        <property
name="datasource">WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB</property>
        <property name="category">large_dataset_optimized</property>
    </properties>
</analytics-record-store>
<!-- The data indexing analyzer implementation -->
<analytics-lucene-analyzer>

<implementation>org.apache.lucene.analysis.standard.StandardAnalyzer</implementation>
</analytics-lucene-analyzer>
<!-- The number of index data replicas the system should keep, for H/A, this should
be at least 1, e.g. the value 0 means
    there aren't any copies of the data -->
<indexReplicationFactor>1</indexReplicationFactor>
<!-- The number of index shards, should be equal or higher to the number of
indexing nodes that is going to be working,
    ideal count being 'number of indexing nodes * [CPU cores used for indexing per
node]' -->
<shardCount>6</shardCount>
<!-- The amount of index data (in bytes) to be processed at a time by a shard index
worker. Minimum value is 1000. -->
<shardIndexRecordBatchSize>20971520</shardIndexRecordBatchSize>
<!-- The interval in milliseconds, which a shard index processing worker thread
will sleep during index processing operations. This setting
    along with the 'shardIndexRecordBatchSize' setting can be used to increase the
final index batched data amount the indexer processes
    at a given time. Usually, higher the batch data amount, higher the throughput
of the indexing operations, but will have a higher latency
    of record insertion to indexing. Minimum value of this is 10, and a maximum
value is 60000 (1 minute). -->
<shardIndexWorkerInterval>1500</shardIndexWorkerInterval>
<!-- The number of index workers to operate in the current node. This basically
results in the number of execution threads created
    to do the indexing operations of the local shards. When this value is
increased, the parallel I/O operations being done on the
    system grows larger. So a system which can handle parallel I/O operation could
increase this, e.g. SSDs. -->
<indexWorkerCount>1</indexWorkerCount>
<!-- Data purging related configuration -->
<analytics-data-purging>
    <!-- Below entry will indicate purging is enable or not. If user wants to enable
data purging for cluster then this property
        need to be enable in all nodes -->
    <purging-enable>false</purging-enable>
    <cron-expression>0 0 * * ?</cron-expression>

```

```
<!-- Tables that need include to purging. Use regex expression to specify the  
table name that need include to purging.-->  
<purge-include-tables>  
    <table>.*</table>  
    <!--<table>.*jmx.*</table>-->  
</purge-include-tables>  
<!-- All records that insert before the specified retention time will be  
eligible to purge -->
```

```

<data-retention-days>365</data-retention-days>
</analytics-data-purging>
</analytics-dataservice-configuration>

```

Removing persisted data

The following two methods can be used to clear all the event data (both processed and unprocessed events) from the database.

- Run the Analytics Data Backup / Restore Tool with the `-deleteTables` argument, specifying the list of event tables for which you want to clear data. For more information, see [Analytics Data Backup / Restore Tool](#).
- Clean your database, and remove the contents in the `<DAS_HOME>/repository/data` directory at the same time.

Cleaning a database while retaining data in the `<DAS_HOME>/repository/data` directory does not result in permanently removing the persisted data.

If you want to clear only index data, see [Configuring Indexes - Removing index data](#).

Implementation With Different Database Types

For detailed information on configuring a record store with each database type, see the following topics.

- [Configuring Data Persistence With Cassandra](#)
- [Configuring Data Persistence With HBase or HDFS](#)
- [Configuring Data Persistence With RDBMS](#)

Configuring Data Persistence With Cassandra

WSO2 DAS supports Apache Cassandra for its underlying Data Access Layer (DAL). In order to use the Cassandra DAL component, a pre-configured installation of Apache Cassandra (2.0.x or 2.1.x version) is required. The Cassandra implementation for the DAS DAL contains the implementations of Analytics Record Store.

The following sections describe both implementations of the Cassandra DAL component.

- [Enabling the Cassandra datasource provider](#)
- [Configurations for the Analytics Record Store](#)
- [Configuring the datasources](#)

The Cassandra datasource does not support pagination or record count at the datasource level. Note that this does not affect the functions of the APIM Analytics, ESB Analytics and IS Analytics dashboards because all the records in these dashboards are indexed, and the record counts are obtained via the Apache Lucene functionality.

Enabling the Cassandra datasource provider

In order to use Cassandra as the record store, the Cassandra datasource provider should be enabled as follows.

1. Open the `<DAS_HOME>/repository/conf/datasources/analytics-datasources.xml` file. The Cassandra datasource provider is commented out by default as shown below.

```

<!--<provider>org.wso2.carbon.datasource.reader.cassandra.CassandraDataSourceRead
er</provider>-->

```

2. Uncomment the Cassandra datasource provider as shown below.

```
<provider>org.wso2.carbon.datasource.reader.cassandra.CassandraDataSourceReader</
provider>
```

Configurations for the Analytics Record Store

To configure Cassandra as the underlying datasource implementation for the Analytics Record Store, specify the Cassandra configurations in the <DAS_HOME>/repository/conf/analytics/analytics-config.xml file as shown in the example below.

```
<analytics-record-store name="EVENT_STORE">

<implementation>org.wso2.carbon.analytics.datasource.cassandra.CassandraAnalyticsRecor
dStore</implementation>
<properties>
    <!-- the data source name mentioned in data sources configuration -->
    <property name="datasource">WSO2_ANALYTICS_DS_CASSANDRA</property>
</properties>
</analytics-record-store>
```

The properties of the above configuration are described below.

Property	Description
<implementation>	The implementation class of the Analytics Record Store relevant for Cassandra, which is carbon.analytics.datasource.cassandra.CassandraAnalyticsRecordStore.
<property name="datasource">	The Carbon datasource name of the CASSANDRA type that is used to find the associated data source.

Configuring the datasources

Change the configurations of the WSO2_ANALYTICS_EVENT_STORE_DB datasource in the <DAS_HOME>/repository/conf/datasources/analytics-datasources.xml file accordingly. For information on the datasource configurations, see [Configuring a Cassandra Datasource](#).

Configuring Data Persistence With HBase or HDFS

WSO2 DAS supports Apache HBase and Apache HDFS for its underlying [Data Access Layer \(DAL\)](#). To use this HBase DAL component, a pre-configured installation of Apache HBase (version 1.1.2 and upwards) running on top of Apache Hadoop (version 2.6.0 and upwards) is required.

The HBase DAL component uses Apache HBase for storing events (**Analytics Record Store**). You have the option of specifying either or both of these implementations in their deployment. For example, you can specify HBase for storing events, and use a relational storage solution for storing index-related information.

The following sections describe both implementations of the HBase DAL component.

- [Enabling the HBase datasource provider](#)
- [Configurations for the Analytics Record Store](#)
- [Configuring the datasources](#)

The HBase datasource does not support pagination or record count. Note that this does not affect the functions of the APIM Analytics, ESB Analytics and IS Analytics dashboards because all the records in

these dashboards are indexed, and the record counts are obtained via the Apache Lucene functionality.

It is required that all HBase/HDFS nodes and all DAS nodes are time synced. Utilities such as [ntpd](#) could be used for this purpose.

Enabling the HBase datasource provider

In order to use HBase as the record store, the HBase datasource provider should be enabled as follows.

1. Open the <DAS_HOME>/repository/conf/datasources/analytics-datasources.xml file. The HBase datasource provider is commented out by default as shown below.

```
<!--<provider>org.wso2.carbon.datasource.reader.hadoop.HBaseDataSourceReader</pro
vider>-->
```

2. Uncomment the HBase datasource provider as shown below.

```
<provider>org.wso2.carbon.datasource.reader.hadoop.HBaseDataSourceReader</provide
r>
```

Configurations for the Analytics Record Store

For configuring HBase as the underlying datasource implementation for the Analytics Record Store, specify the HBase configurations in the <DAS_HOME>/repository/conf/analytics/analytics-config.xml file as shown in the below example.

```
<analytics-record-store name="EVENT_STORE">

    <implementation>org.wso2.carbon.analytics.datasource.hbase.HBaseAnalyticsRecordStore</
implementation>
        <properties>
            <!-- the data source name mentioned in data sources configuration -->
            <property name="datasource">WSO2_ANALYTICS_RS_DB_HBASE</property>
        </properties>
    </analytics-record-store>
```

The properties of the above configurations are described below.

Property	Description
<implementation>	The implementation class of the Analytics Record Store relevant for HBase, which is org.wso2.carbon.analytics.datasource.hbase.HBaseAnalyticsRecordStore.
<property name="datasource">	The Carbon datasource name of the type "HBASE", which is used to look up to find the associated HBase data source.

Configuring the datasources

Change the configurations of the WSO2_ANALYTICS_EVENT_STORE_DB datasource in the <DAS_HOME>/repository/conf/datasources/analytics-datasources.xml file as shown below.

ory/conf/datasources/analytics-datasources.xml file accordingly. For information on the datasource configurations, see [Configuring a HBase Datasource](#).

Configuring Data Persistence With RDBMS

WSO2 DAS supports several RDBMS types for its underlying [Data Access Layer \(DAL\)](#). In order to use the RDBMS DAL component, a pre-configured installation of a RDBMS is required. The RDBMS implementation for the DAS DAL contains the implementations of the Analytics Record Store. The following sections describe both implementations of the Cassandra DAL component

- Enabling the RDBMS datasource provider
- Configurations for the Analytics Record Store
- RDBMS query configuration
- Configuring the datasources

The RDBMS datasource supports both pagination and record count. Pagination support and record count support are disabled by default. You can enable them by setting the value of `paginationSupported` and `recordCountSupported` properties in the `<Das_Home>/repository/conf/analytics/rdbms-conf.xml` file to true.

Enabling the RDBMS datasource provider

In order to use RDBMS as the record store, ensure that the RDBMS datasource provider is enabled by following the steps below.

RDBMS is the default database type of the record store. Therefore, the RDBMS datasource provider is uncommented by default.

1. Open the `<Das_Home>/repository/conf/datasources/analytics-datasources.xml` file.
2. If the datasource provider is commented out, uncomment it as shown below.

```
<provider>org.wso2.carbon.ndatasource.rdbms.RDBMSDataSourceReader</provider>
```

Configurations for the Analytics Record Store

The Analytics Record Store consists of two datasource configurations as follows.

Configuring the Event Store

For configuring RDBMS as the underlying datasource implementation for the Event Store of the Analytics Record Store, specify the RDBMS configurations in the `<Das_Home>/repository/conf/analytics/analytics-conf.xml` file as shown in the example below.

```
<analytics-record-store name="EVENT_STORE">

    <implementation>org.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsRecordStore</implementation>
    <properties>
        <!-- the data source name mentioned in data sources configuration -->
        <property name="datasource">WSO2_ANALYTICS_EVENT_STORE_DB</property>
    </properties>
</analytics-record-store>
```

The properties of the above configuration are described below.

Property	Description
<implementation>	The implementation class of the Analytics Record Store relevant for RDBMS, which is com.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsRecordStore.
<property name="datasource">	The Carbon datasource name of the RDBMS type, that is used to find the associated RDBMS data source.

Configuring the Processed Data Store

To configure RDBMS as the underlying datasource implementation for the Processed Data Store of the Analytics Record Store, specify the RDBMS configurations in the <DAS_HOME>/repository/conf/analytics/analytics-config.xml file as shown in the sample below.

```
<analytics-record-store name = "PROCESSED_DATA_STORE">

<implementation>org.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsRecordStore</implementation>
    <properties>
        <property name="datasource">WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB</property>
    </properties>
</analytics-record-store>
```

The properties of the above configuration are described below.

Property	Description
<implementation>	The implementation class of the Analytics Record Store relevant for RDBMS, which is com.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsRecordStore.
<property name="datasource">	The Carbon datasource name of the RDBMS type, that is used to find the associated RDBMS data source.

RDBMS query configuration

The above Analytics Record Store depends on a query configuration to execute its implementation. The query configuration contains SQL query templates for each of the RDBMS servers that it interfaces with. Using this configuration, you can modify the existing queries for fine tuning, or create new query configurations for a RDBMS that is not configured out of the box. You can find these configurations in the <DAS_HOME>/repository/conf/analytics/rdbms-config.xml file as shown in the example below. Click on the relevant tab to view the query templates for each database category.

```
h2mysql - large_dataset_optimized categorymysql - write_optimizedmysql - read_write_optimizedoracle
Microsoft SQL ServerPostgreSQLSQLDB2.*
```

```

<database name = "h2.*" minVersion = "1.0" maxVersion = "10.0">
    <recordCountSupported>false</recordCountSupported>
    <paginationSupported>true</paginationSupported>
    <paginationMode>MODE1</paginationMode>
    <recordTableCheckQuery>SELECT 1 FROM {{TABLE_NAME}} LIMIT 1</recordTableCheckQuery>
    <recordCountQuery>SELECT COUNT(*) FROM {{TABLE_NAME}} WHERE timestamp >= ? AND timestamp < ?</recordCountQuery>
    <recordDeletionQuery>DELETE FROM {{TABLE_NAME}} WHERE timestamp >= ? AND timestamp < ?</recordDeletionQuery>
    <recordDeletionWithIdsQuery>DELETE FROM {{TABLE_NAME}} WHERE record_id IN ({{RECORD_IDS}})</recordDeletionWithIdsQuery>
    <recordMergeQuery>MERGE INTO {{TABLE_NAME}} (partition_key, timestamp, data, record_id) KEY (record_id) VALUES (?, ?, ?, ?)</recordMergeQuery>
    <recordRetrievalQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}} WHERE partition_key >= ? and partition_key < ? AND timestamp >= ? AND timestamp < ? LIMIT ?,?</recordRetrievalQuery>
    <recordRetrievalWithIdsQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}} WHERE record_id IN ({{RECORD_IDS}})</recordRetrievalWithIdsQuery>
    <recordTableDeleteQueries>
        <query>DROP INDEX IF EXISTS {{TABLE_NAME}}_TIMESTAMP</query>
        <query>DROP INDEX IF EXISTS {{TABLE_NAME}}_PARTITION_KEY</query>
        <query>DROP TABLE IF EXISTS {{TABLE_NAME}}</query>
    </recordTableDeleteQueries>
    <recordTableInitQueries>
        <query>CREATE TABLE {{TABLE_NAME}} (record_id VARCHAR(50), timestamp BIGINT, data BINARY, partition_key INT, PRIMARY KEY(record_id))</query>
        <query>CREATE INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}} (timestamp)</query>
        <query>CREATE INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}} (partition_key)</query>
    </recordTableInitQueries>
</database>

```

```

<database name = "mysql" category = "large_dataset_optimized">
    <recordCountSupported>false</recordCountSupported>
    <paginationSupported>true</paginationSupported>
    <paginationMode>MODE1</paginationMode>
    <recordTableCheckQuery>SELECT 1 FROM {{TABLE_NAME}} LIMIT
1</recordTableCheckQuery>
    <recordCountQuery>SELECT COUNT(*) FROM {{TABLE_NAME}} WHERE timestamp >= ?
AND timestamp < ?</recordCountQuery>
    <recordDeletionQuery>DELETE FROM {{TABLE_NAME}} WHERE timestamp >= ? AND
timestamp < ?</recordDeletionQuery>
    <recordDeletionWithIdsQuery>DELETE FROM {{TABLE_NAME}} WHERE record_id IN
({{RECORD_IDS}})</recordDeletionWithIdsQuery>
    <recordMergeQuery>INSERT INTO {{TABLE_NAME}} (partition_key, timestamp, data,
record_id) VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE
partition_key=VALUES(partition_key), timestamp=VALUES(timestamp),
data=VALUES(data)</recordMergeQuery>
    <forwardOnlyReadEnabled>true</forwardOnlyReadEnabled>
    <fetchSize>-2147483648</fetchSize>
    <recordRetrievalQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}}
WHERE partition_key >= ? and partition_key < ? AND timestamp >= ? AND
timestamp < ? LIMIT ?,?</recordRetrievalQuery>
    <recordRetrievalWithIdsQuery>SELECT record_id, timestamp, data FROM
{{TABLE_NAME}} WHERE record_id IN ({RECORD_IDS})</recordRetrievalWithIdsQuery>
    <recordTableDeleteQueries>
        <query>DROP INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}</query>
        <query>DROP INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}</query>
        <query>DROP TABLE IF EXISTS {{TABLE_NAME}}</query>
    </recordTableDeleteQueries>
    <recordTableInitQueries>
        <query>CREATE TABLE {{TABLE_NAME}} (record_id VARCHAR(50), timestamp
BIGINT, data LONGBLOB, partition_key INT, PRIMARY KEY(record_id))
ENGINE='MyISAM'</query>
        <query>CREATE INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}
(timestamp)</query>
        <query>CREATE INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}
(partition_key)</query>
    </recordTableInitQueries>
</database>

```

```

<database name="mysql" category="write_optimized">
    <recordCountSupported>false</recordCountSupported>
    <paginationSupported>true</paginationSupported>
    <paginationMode>MODE1</paginationMode>
    <recordTableCheckQuery>SELECT 1 FROM {{TABLE_NAME}} LIMIT 1</recordTableCheckQuery>
    <recordCountQuery>SELECT COUNT(*) FROM {{TABLE_NAME}} WHERE timestamp >= ? AND timestamp < ?</recordCountQuery>
    <recordDeletionQuery>DELETE FROM {{TABLE_NAME}} WHERE timestamp >= ? AND timestamp < ?</recordDeletionQuery>
    <recordDeletionWithIdsQuery>DELETE FROM {{TABLE_NAME}} WHERE record_id IN ({{RECORD_IDS}})</recordDeletionWithIdsQuery>
    <recordMergeQuery>INSERT INTO {{TABLE_NAME}} (partition_key, timestamp, data, record_id) VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE partition_key=VALUES(partition_key), timestamp=VALUES(timestamp), data=VALUES(data)</recordMergeQuery>
    <forwardOnlyReadEnabled>true</forwardOnlyReadEnabled>
    <fetchSize>-2147483648</fetchSize>
    <recordRetrievalQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}} WHERE partition_key >= ? and partition_key < ? AND timestamp >= ? AND timestamp < ? LIMIT ?,?</recordRetrievalQuery>
    <recordRetrievalWithIdsQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}} WHERE record_id IN ({{RECORD_IDS}})</recordRetrievalWithIdsQuery>
    <recordTableDeleteQueries>
        <query>DROP INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}</query>
        <query>DROP INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}</query>
        <query>DROP TABLE IF EXISTS {{TABLE_NAME}}</query>
    </recordTableDeleteQueries>
    <recordTableInitQueries>
        <query>CREATE TABLE {{TABLE_NAME}} (record_id VARCHAR(128), timestamp BIGINT, data LONGBLOB, partition_key INT, PRIMARY KEY(record_id)) ENGINE='MyISAM'</query>
        <query>CREATE INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}} (timestamp)</query>
        <query>CREATE INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}} (partition_key)</query>
    </recordTableInitQueries>
</database>

```

```

<database name="mysql" category="read_write_optimized">
    <recordCountSupported>false</recordCountSupported>
    <paginationSupported>true</paginationSupported>
    <paginationMode>MODE1</paginationMode>
    <recordTableCheckQuery>SELECT 1 FROM {{TABLE_NAME}} LIMIT 1</recordTableCheckQuery>
    <recordCountQuery>SELECT COUNT(*) FROM {{TABLE_NAME}} WHERE timestamp >= ? AND timestamp < ?
    </recordCountQuery>
    <recordDeletionQuery>DELETE FROM {{TABLE_NAME}} WHERE timestamp >= ? AND timestamp < ?
    </recordDeletionQuery>
    <recordDeletionWithIdsQuery>DELETE FROM {{TABLE_NAME}} WHERE record_id IN ({{RECORD_IDS}})</recordDeletionWithIdsQuery>
    <recordMergeQuery>INSERT INTO {{TABLE_NAME}} (partition_key, timestamp, data, record_id) VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE partition_key=VALUES(partition_key), timestamp=VALUES(timestamp), data=VALUES(data)</recordMergeQuery>
    <forwardOnlyReadEnabled>true</forwardOnlyReadEnabled>
    <fetchSize>-2147483648</fetchSize>
    <recordRetrievalQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}} WHERE partition_key >= ? and partition_key < ? AND timestamp >= ? AND timestamp < ? LIMIT ?,?</recordRetrievalQuery>
    <recordRetrievalWithIdsQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}} WHERE record_id IN ({{RECORD_IDS}})</recordRetrievalWithIdsQuery>
    <recordTableDeleteQueries>
        <query>DROP INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}</query>
        <query>DROP INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}</query>
        <query>DROP TABLE IF EXISTS {{TABLE_NAME}}</query>
    </recordTableDeleteQueries>
    <recordTableInitQueries>
        <query>CREATE TABLE {{TABLE_NAME}} (record_id VARCHAR(128), timestamp BIGINT, data LONGBLOB, partition_key INT, PRIMARY KEY(record_id)) ENGINE='InnoDB'</query>
        <query>CREATE INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}} (timestamp)</query>
        <query>CREATE INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}} (partition_key)</query>
    </recordTableInitQueries>
</database>

```

```

<database name = "oracle">
    <recordCountSupported>false</recordCountSupported>
    <paginationSupported>true</paginationSupported>
    <paginationMode>MODE2</paginationMode>
    <recordTableCheckQuery>SELECT 1 FROM {{TABLE_NAME}} WHERE
rownum=1</recordTableCheckQuery>
    <recordCountQuery>SELECT COUNT(*) FROM {{TABLE_NAME}} WHERE timestamp >= ?
AND timestamp < ?</recordCountQuery>
    <recordDeletionQuery>DELETE FROM {{TABLE_NAME}} WHERE timestamp >= ? AND
timestamp < ?</recordDeletionQuery>
    <recordDeletionWithIdsQuery>DELETE FROM {{TABLE_NAME}} WHERE record_id IN
({{RECORD_IDS}})</recordDeletionWithIdsQuery>
    <recordInsertQuery>INSERT INTO {{TABLE_NAME}} (partition_key, timestamp, data,
record_id) VALUES (?, ?, ?, ?)</recordInsertQuery>
    <recordUpdateQuery>UPDATE {{TABLE_NAME}} SET partition_key = ?, timestamp = ?, 
data = ? WHERE record_id = ?</recordUpdateQuery>
    <recordMergeQuery>MERGE INTO {{TABLE_NAME}} dest USING( SELECT ?
partition_key, ? timestamp, ? data, ? record_id FROM dual) src ON(dest.record_id =
src.record_id) WHEN NOT MATCHED THEN INSERT(partition_key, timestamp, data, record_id)
VALUES(src.partition_key, src.timestamp, src.data, src.record_id) WHEN MATCHED THEN
UPDATE SET dest.partition_key = src.partition_key, dest.timestamp = src.timestamp,
dest.data = src.data</recordMergeQuery>
    <recordRetrievalQuery>SELECT record_id, timestamp, data from (SELECT rnum
RNUM, record_id, timestamp, data FROM {{TABLE_NAME}} WHERE partition_key >= ? and
partition_key < ? AND timestamp >= ? AND timestamp < ? and rnum <= ?)
where RNUM > ?</recordRetrievalQuery>
    <recordRetrievalWithIdsQuery>SELECT record_id, timestamp, data FROM
{{TABLE_NAME}} WHERE record_id IN ({RECORD_IDS})</recordRetrievalWithIdsQuery>
    <recordTableDeleteQueries>
        <query>DROP INDEX {{TABLE_NAME}}_PARTITION_KEY</query>
        <query>DROP INDEX {{TABLE_NAME}}_TIMESTAMP</query>
        <query>DROP TABLE {{TABLE_NAME}}</query>
    </recordTableDeleteQueries>
    <recordTableInitQueries>
        <query>CREATE TABLE {{TABLE_NAME}} (record_id VARCHAR2(50), timestamp
NUMBER(19), data BLOB, partition_key NUMBER(10), PRIMARY KEY(record_id))</query>
        <query>CREATE INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}
(timestamp)</query>
        <query>CREATE INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}
(partition_key)</query>
    </recordTableInitQueries>
</database>

```

```

<database name = "Microsoft SQL Server">
    <recordCountSupported>false</recordCountSupported>
    <paginationSupported>true</paginationSupported>
    <paginationMode>MODE2</paginationMode>
    <blobLengthRequired>true</blobLengthRequired>
    <recordTableCheckQuery>SELECT TOP 1 1 from
{{TABLE_NAME}}</recordTableCheckQuery>
    <recordCountQuery>SELECT COUNT(*) FROM {{TABLE_NAME}} WHERE timestamp >= ?
AND timestamp < ?</recordCountQuery>
    <recordDeletionQuery>DELETE FROM {{TABLE_NAME}} WHERE timestamp >= ? AND
timestamp < ?</recordDeletionQuery>
    <recordDeletionWithIdsQuery>DELETE FROM {{TABLE_NAME}} WHERE record_id IN
({{RECORD_IDS}})</recordDeletionWithIdsQuery>
    <!--recordMergeQuery>MERGE {{TABLE_NAME}} AS dest USING (SELECT ?, ?, ?, ?) AS
src (partition_key, timestamp, data, record_id) ON (dest.record_id = src.record_id)
WHEN MATCHED THEN UPDATE SET partition_key = src.partition_key, timestamp =
src.timestamp, data = src.data WHEN NOT MATCHED THEN INSERT(partition_key, timestamp,
data, record_id) VALUES (src.partition_key, src.timestamp, src.data,
src.record_id);</recordMergeQuery-->
    <recordInsertQuery>INSERT INTO {{TABLE_NAME}} (partition_key, timestamp, data,
record_id) VALUES (?, ?, ?, ?)</recordInsertQuery>
    <recordUpdateQuery>UPDATE {{TABLE_NAME}} SET partition_key = ?, timestamp = ?,
data = ? WHERE record_id = ?</recordUpdateQuery>
    <recordRetrievalQuery>SELECT record_id, timestamp, data FROM (SELECT
ROW_NUMBER() OVER(ORDER BY record_id) AS rownumber, record_id, timestamp, data FROM
{{TABLE_NAME}} WHERE partition_key >= ? and partition_key < ? AND timestamp
>= ? AND timestamp < ?) AS A WHERE A.rownumber <= ? AND A.rownumber >?
?</recordRetrievalQuery>
    <!--recordRetrievalQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}}
WHERE timestamp >= ? AND timestamp < ? OFFSET ? ROWS FETCH NEXT ? ROWS
ONLY</recordRetrievalQuery-->
    <recordRetrievalWithIdsQuery>SELECT record_id, timestamp, data FROM
{{TABLE_NAME}} WHERE record_id IN ({RECORD_IDS})</recordRetrievalWithIdsQuery>
    <recordTableDeleteQueries>
        <query>DROP INDEX {{TABLE_NAME}}_PARTITION_KEY</query>
        <query>DROP INDEX {{TABLE_NAME}}_TIMESTAMP</query>
        <query>DROP TABLE {{TABLE_NAME}}</query>
    </recordTableDeleteQueries>
    <recordTableInitQueries>
        <query>CREATE TABLE {{TABLE_NAME}} (record_id VARCHAR(50), timestamp
BIGINT, data VARBINARY(max), partition_key INTEGER, PRIMARY KEY(record_id))</query>
        <query>CREATE INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}
(timestamp)</query>
        <query>CREATE INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}
(partition_key)</query>
    </recordTableInitQueries>
</database>

```

```

<database name = "PostgreSQL">
    <recordCountSupported>false</recordCountSupported>
    <paginationSupported>true</paginationSupported>
    <paginationMode>MODE1</paginationMode>
    <recordTableCheckQuery>SELECT 1 FROM {{TABLE_NAME}} LIMIT
1</recordTableCheckQuery>
    <recordCountQuery>SELECT COUNT(*) FROM {{TABLE_NAME}} WHERE timestamp >= ?
AND timestamp < ?</recordCountQuery>
    <recordDeletionQuery>DELETE FROM {{TABLE_NAME}} WHERE timestamp >= ? AND
timestamp < ?</recordDeletionQuery>
    <recordDeletionWithIdsQuery>DELETE FROM {{TABLE_NAME}} WHERE record_id IN
({{RECORD_IDS}})</recordDeletionWithIdsQuery>
    <recordInsertQuery>INSERT INTO {{TABLE_NAME}} (partition_key, timestamp, data,
record_id) VALUES (?, ?, ?, ?)</recordInsertQuery>
    <recordUpdateQuery>UPDATE {{TABLE_NAME}} SET partition_key = ?, timestamp = ?, 
data = ? WHERE record_id = ?</recordUpdateQuery>
    <!--recordMergeQuery>INSERT INTO {{TABLE_NAME}} (partition_key, timestamp,
data, record_id) VALUES (?, ?, ?, ?) ON CONFLICT DO UPDATE
partition_key=VALUES(partition_key), timestamp=VALUES(timestamp),
data=VALUES(data)</recordMergeQuery-->
    <forwardOnlyReadEnabled>true</forwardOnlyReadEnabled>
    <fetchSize>1000</fetchSize>
    <recordRetrievalQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}}
WHERE partition_key >= ? and partition_key < ? AND timestamp >= ? AND
timestamp < ? OFFSET ? LIMIT ?</recordRetrievalQuery>
    <recordRetrievalWithIdsQuery>SELECT record_id, timestamp, data FROM
{{TABLE_NAME}} WHERE record_id IN ({RECORD_IDS})</recordRetrievalWithIdsQuery>
    <recordTableDeleteQueries>
        <query>DROP INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}</query>
        <query>DROP INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}</query>
        <query>DROP TABLE IF EXISTS {{TABLE_NAME}}</query>
    </recordTableDeleteQueries>
    <recordTableInitQueries>
        <query>CREATE TABLE {{TABLE_NAME}} (record_id VARCHAR(50), timestamp
BIGINT, data BYTEA, partition_key INTEGER, PRIMARY KEY(record_id))</query>
        <query>CREATE INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}
(timestamp)</query>
        <query>CREATE INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}
(partition_key)</query>
    </recordTableInitQueries>
</database>

```

```

<database name = "DB2.*">
    <recordCountSupported>false</recordCountSupported>
    <paginationSupported>true</paginationSupported>
    <paginationMode>MODE1</paginationMode>
    <blobLengthRequired>true</blobLengthRequired>
    <recordTableCheckQuery>SELECT 1 FROM {{TABLE_NAME}} FETCH FIRST 1 ROWS
ONLY</recordTableCheckQuery>
    <recordCountQuery>SELECT COUNT(*) FROM {{TABLE_NAME}} WHERE timestamp >= ?
AND timestamp < ?</recordCountQuery>
    <recordDeletionQuery>DELETE FROM {{TABLE_NAME}} WHERE timestamp >= ? AND
timestamp < ?</recordDeletionQuery>
    <recordDeletionWithIdsQuery>DELETE FROM {{TABLE_NAME}} WHERE record_id IN
({{RECORD_IDS}})</recordDeletionWithIdsQuery>
    <recordMergeQuery>MERGE INTO {{TABLE_NAME}} AS dest USING (VALUES(?, ?, ?, ?, ?))
AS src (partition_key, timestamp, data, record_id) ON dest.record_id = src.record_id
WHEN MATCHED THEN UPDATE SET dest.partition_key = src.partition_key, dest.timestamp =
src.timestamp, dest.data = src.data WHEN NOT MATCHED THEN INSERT (partition_key,
timestamp, data, record_id) VALUES (src.partition_key, src.timestamp, src.data,
src.record_id)</recordMergeQuery>
    <recordInsertQuery>INSERT INTO {{TABLE_NAME}} (partition_key, timestamp, data,
record_id) VALUES (?, ?, ?, ?, ?)</recordInsertQuery>
    <recordUpdateQuery>UPDATE {{TABLE_NAME}} SET partition_key = ?, timestamp = ?,
data = ? WHERE record_id = ?</recordUpdateQuery>
    <recordRetrievalQuery>SELECT record_id, timestamp, data FROM {{TABLE_NAME}}
WHERE partition_key >= ? and partition_key < ? AND timestamp >= ? AND
timestamp < ? LIMIT ?,?</recordRetrievalQuery>
    <recordRetrievalWithIdsQuery>SELECT record_id, timestamp, data FROM
{{TABLE_NAME}} WHERE record_id IN ({RECORD_IDS})</recordRetrievalWithIdsQuery>
    <recordTableDeleteQueries>
        <query>DROP INDEX {{TABLE_NAME}}_PARTITION_KEY</query>
        <query>DROP INDEX {{TABLE_NAME}}_TIMESTAMP</query>
        <query>DROP TABLE {{TABLE_NAME}}</query>
    </recordTableDeleteQueries>
    <recordTableInitQueries>
        <query>CREATE TABLE {{TABLE_NAME}} (record_id VARCHAR(50) NOT NULL,
timestamp BIGINT, data BLOB(2G) NOT LOGGED, partition_key INTEGER, PRIMARY
KEY(record_id))</query>
        <query>CREATE INDEX {{TABLE_NAME}}_TIMESTAMP ON {{TABLE_NAME}}
(timestamp)</query>
        <query>CREATE INDEX {{TABLE_NAME}}_PARTITION_KEY ON {{TABLE_NAME}}
(partition_key)</query>
    </recordTableInitQueries>
</database>

```

The above configuration properties are described below.

Property	Description
database name	The target RDBMS name to which this query template applies, a regular expression can be put here, to give a pattern on to which a database product name can be mapped. For example, DB2 will give different product name strings when running in Windows and Unix based OS environments, so a regular expression like "DB2.*" will match for all DB2 based database server environments.

minVersion	The minimum version of the database server this configuration will match to, this will be of format "majorVersion.minorVersion".
maxVersion	The maximum version of the database server this configuration will match to, this will be of format "majorVersion.minorVersion".
recordCountSupported	This property specifies whether it is possible to take a count of all the records in the record store.
paginationSupported	This property specifies whether dividing the output of the record store to manageable chunks is allowed.
paginationMode	This property specifies the pagination mode. Possible values are as follows.
blobLengthRequired	This property specifies whether a length should be assigned to data blocks saved in the record store or not.
recordCountQuery	The query template to take a count of the records in the record store.
recordDeletionQuery	The query template to delete a record in the record store.
recordDeletionWithIdsQuery	The query template to delete records with specific IDs in the record store.
recordMergeQuery	The query template to merge two rows of data of a table in the record store.
forwardOnlyReadEnabled	If this property is set to true, the cursor can only move forward on the result set when retrieving records from record store.
fetchSize	Number of rows that should be fetched from the database if more rows are needed on the generated result set when retrieving records from record store.
recordInsertQuery	The query template to insert new rows of data to a table in the record store.
recordUpdateQuery	The query template to modify the existing table rows in the record store.
recordRetrievalQuery	The query template to retrieve a record from the record store.
recordRetrievalWithIdsQuery	The query template to retrieve records with specific IDs from the record store.
recordTableCheckQuery	The query template to check tables in the record store.
recordTableDeleteQueries	The query template to delete a table in the record store.
recordTableInitQueries	The query template to initialize the tables in the record store.

Configuring the datasources

Change the configurations of the WSO2_ANALYTICS_EVENT_STORE_DB datasource in the <DAS_HOME>/repository/conf/datasources/analytics-datasources.xml file accordingly. For information on the datasource configurations, see [Configuring an RDBMS Datasource](#).

Accessing Persisted Data

In WSO2 DAS, persisted data can be accessed in the same DAS instance by connecting to underlying datasources

via the AnalyticsDataServices OSGi service. This service provides the interface to interact with the event datasource. This section explains how to configure the Analytics API to interact with analytics data and to configure the parameters relating to the connections made to access AnalyticsDataServices OSGI services in remote instances.

Configuring the mode

The data service accessing mode of the node is specified in the <DAS_HOME>/repository/conf/analytics/analytics-data-config.xml file. It can be one of the following.

Mode	Description
LOCAL	The Analytics API only accesses the AnalyticsDataServices OSGI service within itself.
REMOTE	The Analytics API only accesses the AnalyticsDataServices OSGI service in a remote instance. This mode is suitable when the node is a light weight node and does not contain an AnalyticsDataServices OSGI service. When this mode is set, configure the connection related parameters as required.
AUTO	This is the default mode. The Analytics API of a DAS server node always has access to a AnalyticsDataServices OSGi service that exists within that same server node. At the same time, the same API can be used to change the mode and connect to a remote instance. This is done by setting the connection mode to Auto which allows the connection mode to be switched between LOCAL and REMOTE depending on the availability of the required AnalyticsDataServices OSGi service . When this mode is set, configure the connection related parameters as required to connect to remote instances.

Configuring the connection related parameters

The following parameters in the <DAS_HOME>/repository/conf/analytics/analytics-data-config.xml file can be configured to optimize the performance of a node in terms of resource consumption when accessing data services.

These configurations are used only when the data service accessing mode of the node is REMOTE.

Parameter	Description	Default Value
URL	The URL of the server in which the AnalyticsDataService OSGi service is hosted.	http://localhost:9763
Username	The user name to access the server in which the required AnalyticsDataService OSGi service is hosted.	admin
Password	The password to access the server in which the required data services are hosted.	admin
MaxConnections	The maximum number of connections that are allowed to be made from the node to remote instances in order to access the AnalyticsDataService OSGi service.	200
MaxConnectionsPerRoute	The maximum number of connections per route that are allowed to be made from the node to remote instances in order to access the AnalyticsDataService OSGi service.	200

SocketConnectionTimeout	The number of milliseconds after which the socket connection should time out when the node connects to an AnalyticsDataService OSGi service.	60000
ConnectionTimeout	The number of milliseconds after which the connection should time out when the node connects to an AnalyticsDataService OSGi service.	60000
TrustStoreLocation	The path to access the trust store. A trust store is required only if the URL used to access remote data services is in HTTPS protocol. If this parameter is not configured, the trust store configured in the <DAS_HOME>/repository/conf/carbon.xml file is used by default.	repository/resources/s This parameter is comi
TrustStorePassword	The password to access the trust store.	wso2carbon This parameter is comi

Switching Databases without Migrating Data

If you disconnect WSO2 DAS from the database it is currently configured with and connect it to a different database type, the index data stored in DAS may get corrupted if no data migration is carried out. In order to avoid this, reindex the data by following the steps given below.

1. Shut down the WSO2 DAS server.
2. Remove all the index data stored in the <DAS_HOME>/repository/data directory.
3. In the <DAS_HOME>/repository/conf/analytics/local-shard-allocation-config.conf file, change the mode for all the shards from NORMAL to INIT.
4. Restart the WSO2 DAS server.

Related links

[Storing Index Data](#)

Collecting Data

The first step in business activity monitoring is to collect the relevant data you need to analyze. DAS provides data agents that capture information about the messages flowing through the ESB, application server, and other products that use the DAS data publisher. The information is then stored in a data store , where it is optimized for analysis.

The following sections describe how to work with these components to aggregate your data:

- Publishing Data to DAS
- Persisting Data for Batch Analytics
- Configuring DAS to Receive Data
- Configuring Received Data

Publishing Data to DAS

The following topics cover the different ways in which data is sent to WSO2 DAS in the form of events.

- Publishing Data Using Event Simulation
- Publishing Data Using Java Client Through Thrift or Binary
- How to Publish 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 CSV File](#)
 - [Sending Multiple Events Using a Datasource](#)
 - [Sending Multiple Events via XML/JSON](#)

Home > Tools > Event Simulator Help

Event Stream Simulator

Send multiple events

Input Data by File [switch to add configuration for simulate by database](#)

File	Stream Configuration	Action
No file has been uploaded		

Browse... No file selected.

Send single event

Event Stream Name *

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.

Send single event

Event Stream Name *

Stream Attributes

Meta Attributes

- timestamp(*long*) *
- isPowerSaverEnabled(*bool*) *
- sensorId(*int*) *
- sensorName(*string*) *

Correlation Attributes

- longitude(*double*) *
- latitude(*double*) *

Payload Attributes

- humidity(*float*) *
- sensorValue(*double*) *

The fields of the above screen will change depending on the attributes you defined in the event stream.

- 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 CSV File
- Sending Multiple Events Using a Datasource
- Sending Multiple Events via XML/JSON

Sending Multiple Events Using a CSV 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.

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.

```
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
```

Follow the steps below to send multiple events to a defined event stream using a CSV file via the event simulator.

- Log in to the product management console using the following URL: https://<PRODUCT_HOST>:<PRODUCT_PORT>/carbon/
- Click **Tools**, and then click **Event Simulator**.
- 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.

Send multiple events

Input Data by File switch to configure database for simulation			
File	Stream Configuration	Delay between events(ms)	Action
events.csv	click the configure button	click the configure button	Configure Delete
Choose File No file chosen	upload		

6. Configure the CSV file by entering the field delimiter as shown below.

Event Mapping Configuration

File name	events.csv
Select the target event stream*	test:1.0.0
Field delimiter*	
Delay between events in milliseconds*	1000
Configure	

7. Click **Play**, to simulate the event flow.

Click the corresponding **Delete** button to, delete the uploaded CSV file along with its configurations.

Send multiple events

Input Data by File switch to configure database for simulation			
File	Stream Configuration	Delay between events(ms)	Action
events.csv	test:1.0.0	click the configure button	Play Configure Delete
Choose File No file chosen	upload		

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**.

Send multiple events

Input Data by File switch to configure database for simulation			
File	Stream Configuration	Delay between events(ms)	Action
events.csv	test:1.0.0	click the configure button	Play Configure Delete
Choose File No file chosen	upload		

6. Enter the datasource information to be used when simulating the events, as shown below.

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.

Send multiple events

Name	Data Source Name	Table Name	Column names	Stream ID	Stream Attributes	Delay between events(ms)	Action
There are no data source configurations							

Configuration Name*: testConfiguration
Data Source Type*: RDBMS
Data Source Name*: MySQLDataSource
Table Name*: testTable
Delay between events in milliseconds*: 1000
Event Stream Name*: test:1.0.0

Map Stream Attributes with DataBase Fields

Table column name	
Payload Attributes	
attrib1(int) *	54
attrib2(string) *	hostname

Test Connection **Save**

- 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.

Name	Data Source Name	Table Name	Column names	Stream ID	Stream Attributes	Action
MySQLDataSourceToSimulate	MySQLDataSource	testTableForCEP	timestamp,powerSaver,id,name,longitude,latitude,humidity,value	org.wso2.event.sensor.stream:1.0.0	timestamp,isPowerSaverEnabled,sensorId,sensorName,longitude,latitude,humidity,sensorValue	Play Delete

Sending Multiple Events via XML/JSON

Follow the procedure below to send multiple events to WSO2 DAS via XML/JSON.

- 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
- 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 individual events.

For detailed instructions to define JSON mapping, see [Input Mapping Types - JSON input mapping](#).

Publishing Data Using Java Client Through Thrift or Binary

- 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 saved in Cassandra with the `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 saved in Cassandra with the `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 Cassandra with the `payload_` 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.commons.version}` refers to the version of the carbon-commons github repository - <https://github.com/wso2/carbon-commons/>. 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`
- `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`

Adding dependencies using POM file

Alternatively, add the following Maven project dependency entries to your POM file. Note that `${carbon.commons.version}` refers to the version of the carbon-commons github repository - <https://github.com/wso2/carbon-commons/>. It is always recommended to use the dependency entry from the latest released version.

Maven repository

```
<repositories>
    <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>
```

Maven pom dependency

```
<dependency>
    <groupId>org.wso2.carbon.analytics-common</groupId>
    <artifactId>org.wso2.carbon.databridge.agent</artifactId>
    <version>${carbon.common.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.carbon.analytics-common</groupId>
    <artifactId>org.wso2.carbon.databridge.commons</artifactId>
    <version>${carbon.common.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.carbon.analytics-common</groupId>
    <artifactId>org.wso2.carbon.databridge.commons.thrift</artifactId>
    <version>${carbon.common.version}</version>
</dependency>
```

Specify the relevant version for the `<version>` element.

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>wso2carbon</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>wso2carbon</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>

```

To configure the above parameters in the <DAS_HOME>/repository/conf/data-bridge/data-agent-conf.xml file in order to tune performance, follow the instructions in [Performance Tuning](#).

2. Instantiate the data publisher as follows: AgentHolder.setConfigPath("/path/to/data/agent/conf.xml")

3. Instantiate and use the data publisher using one of the following configurations:

- `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](#). And similarly if you are passing an receiverURLSet as `tcp://localhost:7611|tcp://localhost:7612|tcp://localhost:7613`, then the corresponding receiverURL 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 `<Das_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 `<Das_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.

```
DataPublisher dataPublisher = new DataPublisher(String type, String receiverURLSet, String authURLSet, String username, String password)
```

Data publisher sample

As a prerequisite for this sample, you need to define the streams in the receiver server (WSO2 DAS/CEP). For information on defining event streams, see [Understanding Event Streams and Event Tables](#).

Follow the procedure below to use the data publisher.

1. Initialize the data publisher as follows.

```
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.

```
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.

```
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.

```
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 milliseconds. 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.

```
Event event = new Event(streamId, System.currentTimeMillis(), new
Object[]{"external"}, null,
new Object[]{aLog});
dataPublisher.tryPublish(event, 100);
```

When you use the `tryPublish` API, it is important to check the value it returns. If it returns `false` (i.e indicating that the event was not sent), you should slow down the process of sending it requests in order to avoid overusing the CPU resources and blocking them for other operations. In order to do this, use the sleep option or similar.

For more information on the usage of data publishers, see the sample in the `<Das_Home>/samples/httpd-logs/` 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.

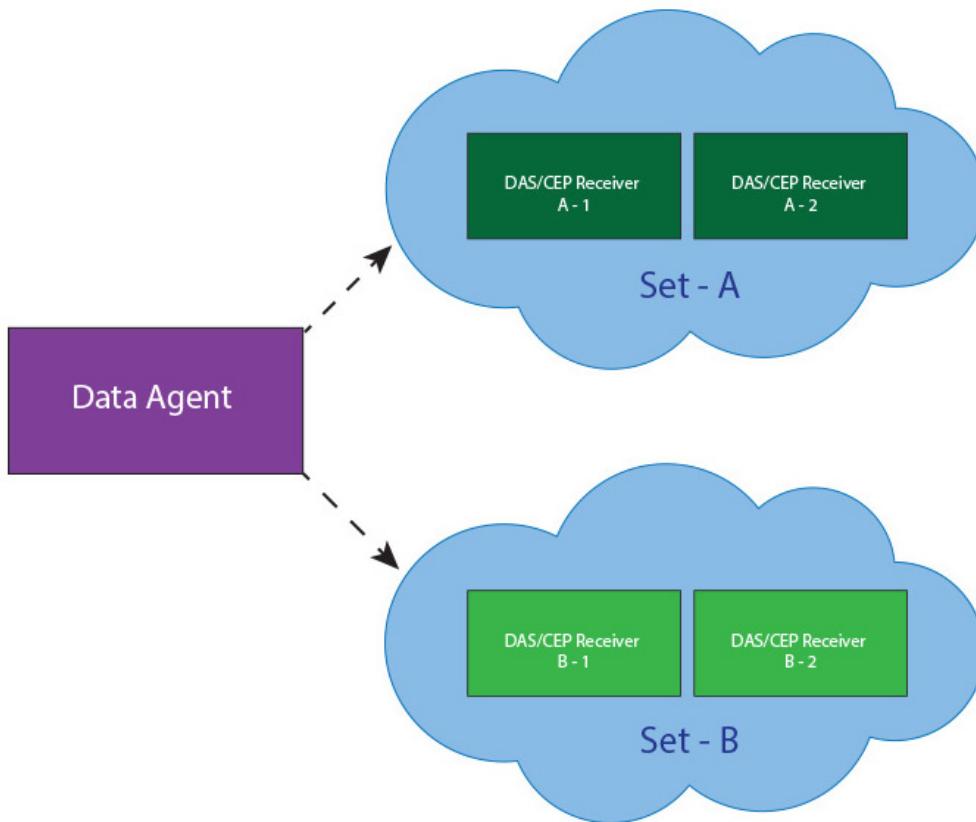
All the scenarios described below are different ways to use Data Agents with multiple receivers using the load balancing functionality. Each approach has its own advantages. Select the most appropriate scenario depending on your requirements.

- 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.

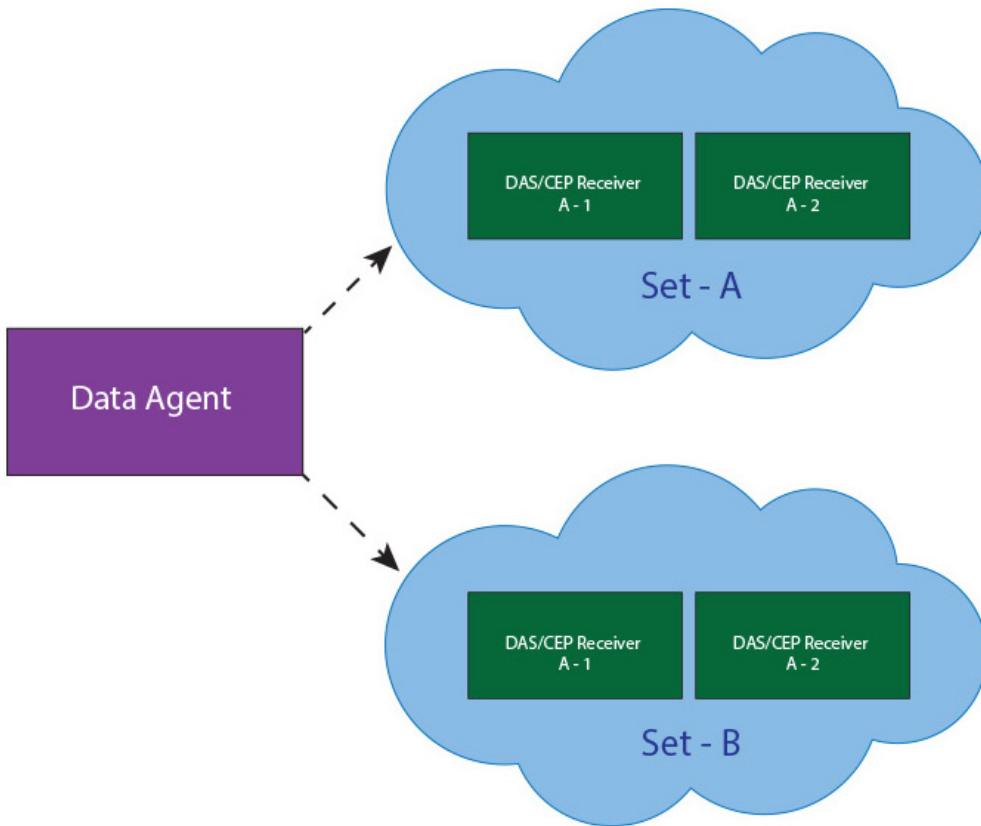
For this functionality, include the server URL in the Data Agent as a general DAS/CEP receiver URL. The URL should be entered in a comma separated format as shown below.

```
Receiver URL = tcp://<DAS/CEP Receiver -1>:<port>,tcp://<DAS/CEP Receiver -2>:<port>,tcp://<DAS/CEP Receiver -3>:<port>
```

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.

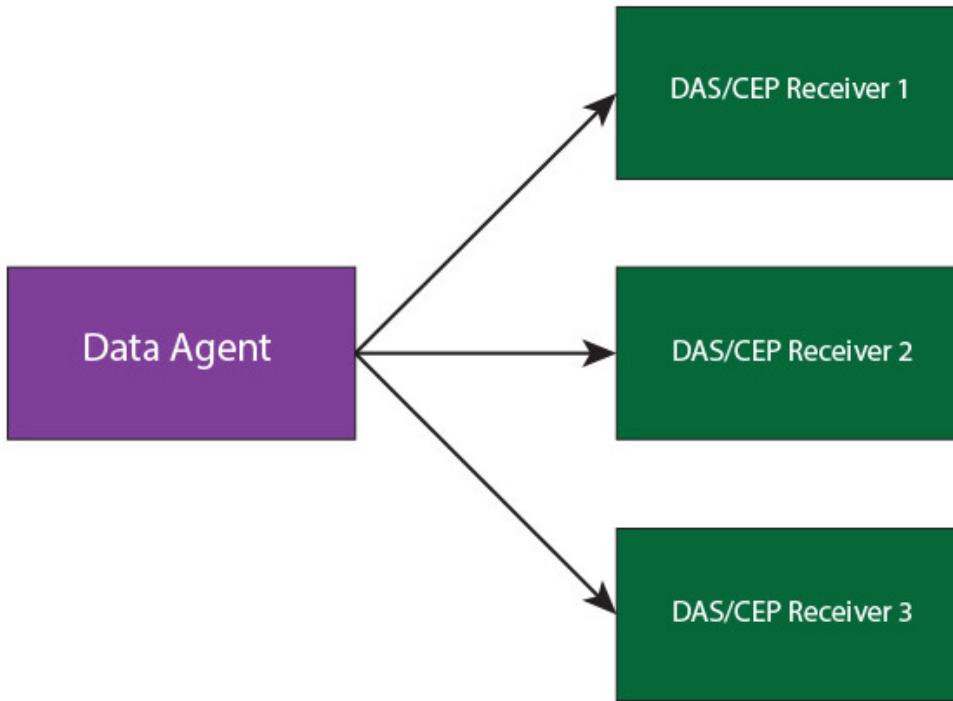
Similar to the other scenarios, you can describe this as a receiver URL. The sets should be mentioned within curly braces separated by commas. Further more, each receiver that belongs to the set, should be within the curly braces and with the receiver URLs in a comma separated format. The receiver URL format is given below.

```
Receiver URL = {tcp://<DAS/CEP Receiver -A-1>:<port>, tcp://<DAS/CEP Receiver -A-2>:<port>},{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 CEP. You can use this functionality to publish the same event to both DAS and CEP servers at the same time. This is useful to perform real time analytics with CEP, to persist the data, and also to perform complex analysis with DAS in nearly real time with the same data.

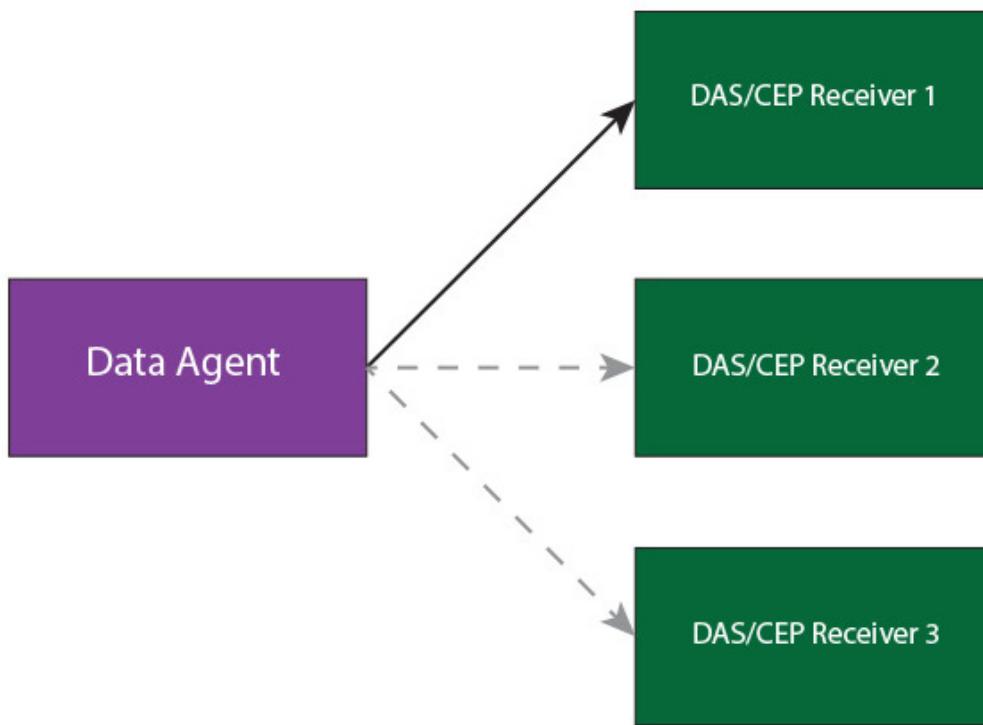
Similar to [load balancing between a set of servers](#), in this scenario you need to modify the Data Agent URL. You should include all DAS/CEP receiver URLs within curly braces ({}) separated with commas as shown below.

```
Receiver URL = {tcp://<DAS/CEP Receiver -1>:<port>} , {tcp://<DAS/CEP Receiver -2>:<port>} , {tcp://<DAS/CEP Receiver -3>:<port>}
```

<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, the events are first sent to DAS/CEP Receiver-1. If it is unavailable, then events will be sent to DAS/CEP Receiver-2. If DAS/CEP Receiver-2 is also unavailable, then the 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.

```

Receiver URL = tcp://<DAS/CEP Receiver -1>:<port>|tcp://<DAS/CEP Receiver
-2>:<port>|tcp://<DAS/CEP Receiver -3>:<port>
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>

How to Publish 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 `<DAS_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.

- [Uploading the C-App](#)
- [Adding the default JMX profile](#)
- [Adding a JMX server profile](#)
- [Viewing the output](#)

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 [Packaging Artifacts as a C-App Archive](#).

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>/capps/JMX_Agent.car` file as shown below.

[Home > Manage > Carbon Applications > Add](#)

Add Carbon Applications

Upload Carbon Application

Carbon Application Artifact(.car)

4. Click **Main**, then click **Carbon Applications**, and then click **List view**, to see the uploaded Carbon application as shown below.

Carbon Applications List

3 Running Carbon Applications.

Carbon Applications	Version	Actions	
Smart_Home_Sample_CApp	1.0.0	Delete	Download
APIM_Realtime_Analytics	1.0.0	Delete	Download
JMX_Agent_CApp	1.0.0	Delete	Download

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

[Add JMX Server Profile](#)

[Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux \(CPU/Memory/OS\)](#)

Profile	Version	Actions
toolbox	1.0.0	Enable Edit Delete

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.

[Edit Profile – toolbox \(Current version:1.0.0\)](#)

All the fields marked with * are mandatory

Basic Information

Schedule*: (?) Cron expression for task scheduling.

JMX Server Information

Server URL*: (?) Enter the monitoring URL for the JMX server.

User Name*: (?) Enter the user name for the JMX server.

Password*: (?) Enter the password for the JMX server.

Attributes: [Add More](#)

MBean	Attribute	Alias	Actions
java.lang:type=Memory	HeapMemoryUsage - init	heap_mem_init	
java.lang:type=Memory	HeapMemoryUsage - max	heap_mem_max	
java.lang:type=Memory	HeapMemoryUsage - used	heap_mem_used	
java.lang:type=Memory	HeapMemoryUsage - committed	heap_mem_committed	
java.lang:type=Memory	NonHeapMemoryUsage - init	non_heap_mem_init	
java.lang:type=Memory	NonHeapMemoryUsage - max	non_heap_mem_max	
java.lang:type=Memory	NonHeapMemoryUsage - used	non_heap_mem_used	
java.lang:type=Memory	NonHeapMemoryUsage - committed	non_heap_mem_committed	
java.lang:type=OperatingSystem	ProcessCpuTime	processCpuTime	
java.lang:type=ClassLoading	LoadedClassCount	loadedClassCount	
java.lang:type=Threading	ThreadCount	threadCount	
java.lang:type=Threading	PeakThreadCount	peakThreadCount	

[Save](#) [Cancel](#)

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.

Select attributes:

Selected Attributes:			
MBean	Attribute	Alias	Actions
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/ /localhost /host/outputwebsocket,j2eeType=Servlet,name=default	eventProvider	Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/ /localhost	Remove
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/ /localhost /host/outputwebsocket,j2eeType=Servlet,name=default	maxTime	Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/ /localhost	Remove

Save **Cancel**

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.

JMX Monitoring Profiles

[Add JMX Server Profile](#)

[Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux \(CPU/Memory/OS\)](#)

Profile	Version	Actions
toolbox	2.0.0	Disable Edit Delete

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

Basic Information

Enter basic JMX profile information.

Name:*

testProfile

Schedule:*

once every 2 seconds (0/2 * * ? * *)

JMX Server Information

Server URL:*

service:jmx:rmi://localhost:11111/jndi/rm

Enter the monitoring URL for the JMX server.

User Name:*

admin

Enter the user name for the JMX server.

Password:*

.....

Enter the password for the JMX server.

The details you enter in the above screen are described below.

Field	Description	Example
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**?**)
Server URL	The JMX server URL.	<p>service:jmx:rmi://localhost:11111/jndi/rmi://localhost:9999 (Use this example to monitor DAS by itself).</p> <p>Change all occurrences of the host and port (if you have set a port offset server) accordingly in the JMX server URL.</p>

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.

```
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/j2eeType=Filter,name=CharsetFilter
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/j2eeType=Filter,name=Tomcat WebSocket (JSR356) Filter
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/j2eeType=Servlet,name=bridgeservlet
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/j2eeType=Servlet,name=default
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/j2eeType=Servlet,name=jsp
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/analytic,j2eeType=Filter,name=Tomcat WebSocket (JSR356) Filter
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/analytic,j2eeType=Servlet,name=cxf
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/analytic,j2eeType=Servlet,name=default
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/analytic,j2eeType=Servlet,name=jsp
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/inputwebsocket,j2eeType=Filter,name=Tomcat WebSocket (JSR356) Filter
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/inputwebsocket,j2eeType=Servlet,name=default
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/inputwebsocket,j2eeType=Servlet,name=jsp
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputputui,j2eeType=Filter,name=Tomcat WebSocket (JSR356) Filter
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputputui,j2eeType=Servlet,name=JAXServlet
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputputui,j2eeType=Servlet,name=default
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputputui,j2eeType=Servlet,name=jsp
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputwebsocket,j2eeType=Filter,name=Tomcat WebSocket (JSR356) Filter
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputwebsocket,j2eeType=Servlet,name=default
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputwebsocket,j2eeType=Servlet,name=jsp
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/portal,j2eeType=Filter,name=JaggeryFilter
```

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.

Select attributes:

Selected Attributes:			
MBean	Attribute	Alias	Actions
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputwebsocket,j2eeType=Servlet,name=default	eventProvider	Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost	<input type="button" value="Remove"/>
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputwebsocket,j2eeType=Servlet,name=default	maxTime	Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost	<input type="button" value="Remove"/>

8. 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

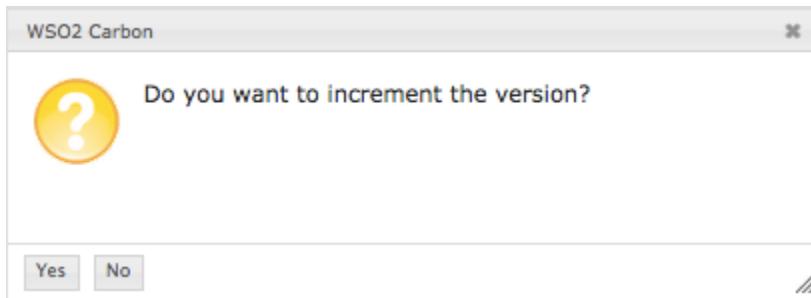
[Add JMX Server Profile](#)

[Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux \(CPU/Memory/OS\)](#)

Profile	Version	Actions
testProfile	1.0.0	

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.



JMX Monitoring Profiles

[Add JMX Server Profile](#)

[Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux \(CPU/Memory/OS\)](#)

Profile	Version	Actions
testProfile	2.0.0	Enable Edit Delete
toolbox	1.0.0	Enable Edit Delete

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_TOOLBLX** for the **Table Name** as shown below.

[Home > Manage > Interactive Analytics > Data Explorer](#)

Data Explorer

Search	
Table Name*	<input type="text" value="JMX_AGENT_TOOLBOX"/>
<input type="button" value="Search"/> <input type="button" value="Reset"/>	
<input type="radio"/> By Date Range <input type="radio"/> By Query	
<input type="button" value="Schedule Data Purging"/>	

4. Click **Search**. You view the published data as shown below.

JMX_AGENT_TOOLBOX										
	meta_clientType	meta_host	heap_mem_init	heap_mem_max	heap_mem_used	heap_mem_committed	non_heap_mem_init	non_heap_mem_max	non_heap_mem_used	non_heap_mem_committed
externalEvent	localhost:11111	268435456	954728448	424374160	802160640	24576000	318767104	192721400	271384576	
externalEvent	localhost:11111	268435456	954728448	427987264	802160640	24576000	318767104	192736968	271384576	
externalEvent	localhost:11111	268435456	954728448	467883152	802160640	24576000	318767104	192931320	271384576	
externalEvent	localhost:11111	268435456	954728448	484188352	802160640	24576000	318767104	192969656	271384576	
externalEvent	localhost:11111	268435456	954728448	505199240	802160640	24576000	318767104	192987064	271384576	
externalEvent	localhost:11111	268435456	954728448	531466808	802160640	24576000	318767104	192990616	271384576	
externalEvent	localhost:11111	268435456	954728448	553166264	802160640	24576000	318767104	192892376	271384576	
externalEvent	localhost:11111	268435456	954728448	584100440	802160640	24576000	318767104	192943304	271384576	

Persisting Data for Batch Analytics

After creating an event stream you can persist it by creating a corresponding table in the WSO2 Data Access Layer. Follow the steps below to persist an event stream.

1. Log in to the management console using the following URL: `https://<DAS_HOST>:<DAS_PORT>/carbon`
2. Click **Main**, and then click **Streams**.
3. Click **Edit** of the corresponding event stream which you want to persist.
4. Click **Next [Persist Event]**.
5. Select the **Persist Event Stream** check box to allow the events in the stream to be persisted.
6. In the **Record Store** field, select **EVENT_STORE** from the list.
7. For each of the attribute types, do the following as required to define the schema of the event stream as shown below.

Home Help

Edit Event Stream

Enter Event Stream Details												
<input checked="" type="checkbox"/> Persist Event Stream												
Record Store EVENT_STORE												
Meta Data Attributes												
<input type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet						
<input type="checkbox"/>	subnet-mask	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
Correlation Data Attributes												
<input type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet						
<input type="checkbox"/>	network-ip	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
Payload Data Attributes												
<input checked="" type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet						
<input checked="" type="checkbox"/>	hostname	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>						
<input checked="" type="checkbox"/>	ip-address	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>						
<input checked="" type="checkbox"/>	state	BOOLEAN	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
Arbitrary Data Attributes												
No arbitrary data attributes are defined												
Attribute Name :		Attribute Type :	INTEGER	Primary Key :	<input type="checkbox"/>	Index Column :	<input type="checkbox"/>	Score Param :	<input type="checkbox"/>	Is a Facet :	<input type="checkbox"/>	<input type="button" value="Add"/>
Advanced												
<input type="button" value="Back"/> <input type="button" value="Save Event Stream"/>												

- Select **Persist Attribute**, if you want to persist a particular attribute.
- Select **Primary Key**, to define an attribute type as a primary key.
- Select **Index Column**, to enable an attribute type to be applied in searches.
- Select **Score Param**, to define an attribute as a score parameter. For more information on score parameters, see [Searching for Data](#).
- Select the **Is a Facet** check box if you want to persist an attribute as a facet. For more information on facets, see [Searching for Data](#).
- Define and add any **Arbitrary Data Attributes** which you want to persist.

8. Click **Save Event Stream** to persist the event stream.

Once an event stream is persisted, an event sink configuration is created for it in XML format. This configuration is saved in the `DAS_HOME>/repository/deployment/server/eventsink` directory.

Configuring DAS to Receive Data

The following topics cover the different ways in which WSO2 DAS 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
- SOAP Event Receiver
- WebSocket Event Receiver
- WebSocket Local Event Receiver
- WSO2Event Event Receiver

Event receiver configuration

An event receiver configuration has four main sections as shown in the example below.

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name* <input type="text" value=""/> (Enter a unique name to identify Event Receiver)	From <input type="text" value="wso2event"/> Select the type of Adapter to receive events
Adapter Properties <input checked="" type="checkbox" value="false"/> Is events duplicated in cluster	
To <input type="text" value="testEventStream:1.0.0"/> The event stream that will be generated by the received events	
Mapping Configuration <input type="text" value="wso2event"/> Select the input message format	
Advanced	

[Add 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.

Section	Description
From	An input event adapter (transport) configuration via which the event receiver receives events.
Adapter properties	Specific properties of the selected input event adapter. For information on configuring adapter properties of various transport types, see Event Receiver Types .
To	The event stream from which the event receiver will fetch the events for processing.
Mapping configuration	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 .

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 WOS2 CEP/DAS.

- To create an event receiver via the management console, you need to have at least one event stream defined.
- Once an event receiver is created, its XML configuration is stored in the <DAS_HOME>/repository/deployment/server/eventreceivers directory.

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.

Home > Manage > Event Processor > Event Receivers Help

Available Event Receivers

[Add Event Receiver](#)

1 Active Event Receivers. 0 Inactive Event Receivers (All)

Event Receiver Name	Message Format	Input Event Adapter Type	Input Stream ID	Actions
TestEventReceiver	xml	http	org.wso2.test:1.0.0	Enable Statistics Enable Tracing Delete Edit

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.

In the following configuration, specify the respective adapter properties based on the transport type of the receiver within the `<from>` element. For the respective adapter properties of the event receiver configuration based on the transport type, see [Event Receiver Types](#).

```

<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.

Adapter property	Description
------------------	-------------

name	Name of the event receiver
statistics	Whether monitoring event statistics is enabled for the receiver
trace	Whether tracing events is enabled for the receiver
xmlns	XML namespace for event receivers
eventAdapterType	Type of the event adapter.
customMapping	Whether a custom mapping is enabled on the receiver.
type	Type of the enabled custom mapping.
streamName	Name of the event stream to which the receiver is mapped.

2. Add the XML file to the <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 **Inactive Event Receivers** in the **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.

[Home](#) > [Manage](#) > [Event Processor](#) > [Event Receivers](#)

[Help](#)

Inactive Event Receivers

File Name	Reason for being inactive	Actions
MQTTTestReceiver.xml	Deployment exception: org/wso2/carbon/event/input/adapter/mqtt/internal/util/MQTTAdapterListener	Delete Source View

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.

For more information on monitoring event statistics of event receivers, see [Event Statistics](#).

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.

For more information on monitoring event statistics of event receivers, see [Event Tracer](#).

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 `<DAS_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.

Property Key	Description	Data Type	Default Value
<code>moveToFolderName</code>	The name of the folder in which the events received should be saved. If a folder with the given name does not already exist in the email server, it will be automatically created when events are received by email-receivers.	String	<code>readMails</code>

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

Enter Event Receiver Details

Event Receiver Name*	EmailInputEventAdapter ⑦ Enter a unique name to identify Event Receiver
From	
Input Event Adapter Type*	email ⑦ Select the type of Adapter to receive events
Adapter Properties	
Receiving Mail Address *	receiver@gmail.com ⑦ The mail address from which this service should fetch incoming mails.
User Name *	receiver ⑦ Mail username
Password *	***** ⑦ Mail Password
Subject *	Email Input Event Adapter Test ⑦ Mails which are coming with above subject will be only processed. (Also need to be plain text format)
Mail Protocol Host (eg: pop.gmail.com or imap.gmail.com) *	pop.gmail.com ⑦ Mail receiver host address
Mail Protocol Port (eg: 995 or 993) *	995 ⑦ Mail receiver protocol
Mail Protocol	pop3 ⑦ The mail protocol to be used to receive messages.
Poll Interval (in seconds)*	10 ⑦ A positive integer
To	
Event Stream*	testEventStream:1.0.0 ⑦ The event stream that will be generated by the received events
Mapping Configuration	
Message Format*	xml ⑦ Select the input message format
+ Advanced	
<input type="button" value="Add 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** that 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="EmailInputEventAdapter" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventreceiver">
<from eventAdapterType="email">
    <property name="transport.PollInterval">10</property>
    <property name="mail.protocol.host">pop.gmail.com</property>
    <property name="email.in.subject">Test Email Input Event Adapter</property>
    <property encrypted="true" name="mail.protocol.password">iLRJ++ICiwFSC+Ji+eOgVEDyBqfAODTQnLzkeVFbeU2cGkokBvHAn/gY
yN3A/98GgZvCMaIod+H6A9A+4wzNkrLNbjUc4IL71kYaairloX2i/QnCxIMcFrzYawj/Jn6z0m9VAsnb6GgIr+
mN1CIZaLZPZw0Hjd9whpVwAkNMtrs=</property>
    <property name="mail.protocol.user">receiver</property>
    <property name="transport.mail.Address">receiver@gmail.com</property>
    <property name="transport.mail.Protocol">pop3</property>
    <property name="mail.protocol.port">995</property>
</from>
.....
<eventReceiver/>

```

The above adapter properties are described below.

Adapter property	Description	Configuration file property	Example
Receiving Mail Address	A valid mail address from which this service should fetch incoming mails.	transport.mail.Address	receiver@gmail.com
User Name	Username of the receiver email account.	mail.protocol.user	test-user
Password	Password of the receiver email account.	mail.protocol.password	test-password
Subject	Only the mails with this subject mentioned will be processed. (The mail should be in plain text format.)	email.in.subject	Email Input Event Adapter Test
Mail Protocol Host	Host address of the mail receiver.	mail.protocol.host	pop.gmail.com/imap.gmail.com
Mail Protocol Port	Port of the mail receiver.	mail.protocol.port	995/993
Mail Protocol	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.	transport.mail.Protocol	pop3/imap
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 <DAS_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.

Property Key	Description	Data Type	Default Value
events.duplicated.in.cluster	If this property is set to true, events received by file-tail receivers are re-created in every node in the cluster.	Boolean	false

Configuring adapter properties

Specify the **Adapter Properties**, when creating a file-tail 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

Input Event Adapter Type* Select the type of Adapter to receive events

Adapter Properties

File path*	/Users/Desktop/abc.txt <small>Absolute path of the file (Eg: /home/cep/cep_3.1.0/wso2cep-3.1.0/repository/logs/wso2carbon.log)</small>
Delay	10 <small>The delay between checks for new content on file in milliseconds.</small>
Start From End*	true <small>Set to true to tail from the end of the file, false to tail from the beginning of the file.</small>

To

Event Stream* The event stream that will be generated by the received events

Mapping Configuration

Message Format* Select the input message format

[Advanced](#)

[Add 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** that 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="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.

Adapter property	Description	Configuration file property	Example
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 `<PRODUCT_HOME>/repository/conf/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.

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 size of the queue that is used to hold events before they are forwarded to the event stream.	Integer	10000

Configuring adapter properties

Specify the **Adapter Properties**, when creating a HTTP event receiver using the management console as shown below.

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="httpReceiver"/> ? <i>Enter a unique name to identify Event Receiver</i>	
From	
Input Event Adapter Type* <input type="text" value="http"/> ? <i>Select the type of Adapter to receive events</i>	
<p>Following url formats are used to receive events For super tenants: <a href="http://localhost:9764/endpoints/<event_receiver_name>">http://localhost:9764/endpoints/<event_receiver_name> <a href="https://localhost:9444/endpoints/<event_receiver_name>">https://localhost:9444/endpoints/<event_receiver_name></p>	
Usage Tips <p>For other tenants: <a href="http://localhost:9764/endpoints/t/<tenant_domain>/<event_receiver_name>">http://localhost:9764/endpoints/t/<tenant_domain>/<event_receiver_name> <a href="https://localhost:9444/endpoints/t/<tenant_domain>/<event_receiver_name>">https://localhost:9444/endpoints/t/<tenant_domain>/<event_receiver_name></p>	
Adapter Properties	
Transport(s)* <input type="text" value="http"/> ?	
To	
Event Stream* <input type="text" value="TestStream:1.0.0"/> ? <i>The event stream that will be generated by the received events</i>	
Mapping Configuration	
Message Format* <input type="text" value="xml"/> ? <i>Select the input message format</i>	
+ Advanced	
<input type="button" value="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** that 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="httpReceiver" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventreceiver">
    <from eventAdapterType="http">
        <property name="transports">https</property>
    </from>
    .....
</eventReceiver>

```

The above adapter properties are described below.

Adapter property	Description	Configuration file property	Example
Transport(s)	Transport type via which the events are received.	transports	https

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 Query in Persistence Mode
- Sample 0503 - Processing a Window Query in High Availability Mode

JMS Event Receiver

JMS event receivers 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 `<Das_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.

Property Key	Description	Data Type	Default Value
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 size of the queue that is used to hold events before they are forwarded to the event stream.	Integer	10000

The following sections describe how to configure a few common JMS event receiver types.

- ActiveMQ Event Receiver
- IBM WebSphere MQ JMS Event Receiver
- Qpid 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](#).

2. Add the following ActiveMQ JMS-specific JAR files to <DAS_HOME>/repository/components/lib/ directory.
 - <ACTIVE MQ_HOME>/lib/activemq-core-xxx.jar
 - <ACTIVE MQ_HOME>/lib/geronimo-j2ee-management_1.1_spec-1.0.1.jar
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](#).

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

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="ActiveMQJMSInputEventAdapter"/> <small>⑦ Enter a unique name to identify Event Receiver</small>	From <input type="text" value="jms"/> <small>⑦ Select the type of Adapter to receive events</small>
Adapter Properties	
Topic/Queue Name* <input type="text" value="Test Topic"/> <small>⑦ Topic/Queue name of the input stream</small>	JNDI Initial Context Factory Class* <input type="text" value="org.apache.activemq.jndi.ActiveMQInitialContextFactory"/> <small>⑦ JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</small>
JNDI Provider URL* <input type="text" value="tcp://localhost:61616"/> <small>⑦ URL of the JNDI provider.</small>	The JMS connection password <input type="text"/>
The JMS connection username <input type="text"/>	Connection Factory JNDI Name* <input type="text" value="TopicConnectionFactory"/> <small>⑦ The JNDI name of the connection factory.</small>
Destination Type* <input type="text" value="topic"/> <small>⑦ Type of the destination.</small>	Enable Durable Subscription <input type="text" value="true"/> <small>⑦ Whether the subscription is durable or not.</small>
Durable Subscriber Client ID <input type="text" value="wso2dasclient1"/> <small>⑦ Name/ID of the durable subscriber (If any value added, Durable subscription will be enabled).</small>	
JMS Properties <input type="text"/> <small>⑦ Axis2 JMS Properties, e.g. "property1: value1, property2: value2"</small>	
To	
Event Stream* <input type="text" value="TestStream:1.0.0"/> <small>⑦ The event stream that will be generated by the received events</small>	
Mapping Configuration	
Message Format* <input type="text" value="xml"/> <small>⑦ Select the input message format</small>	
 Advanced	

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** that 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.ActiveMQInitialContextFact
ory</property>
            <property name="java.naming.provider.url">tcp://localhost:61616</property>
            <property name="transport.jms.DestinationType">topic</property>
            <property name="transport.jms.SubscriptionDurable">true</property>
            <property name="transport.jms.Destination">Test Topic</property>
            <property
name="transport.jms.DurableSubscriberClientID">wso2dasclient1</property>
            <property
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
        </from>
        .....
    </eventReceiver>

```

The above adapter properties are described below.

Adapter Property	Description	Configuration file property
Topic/Queue Name	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.	transport.jms.Destination
JNDI Initial Context Factory Class	JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.	java.naming.factory.initial
J N D I Provider URL	URL of the JNDI provider.	java.naming.provider.url
The JMS connection password	A valid password for the JMS connection.	transport.jms.Password
The JMS connection username	A valid username for the JMS connection.	transport.jms.UserName
Connection Factory JNDI Name	The JNDI name of the connection factory.	transport.jms.ConnectionFactoryJNDIName
Destination Type	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 string of characters to denote a valid id for client connecting as the durable subscriber. (It will enable durable subscription if you add any value here).	transport.jms.DurableSubscriberClient
J M S 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 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:

```

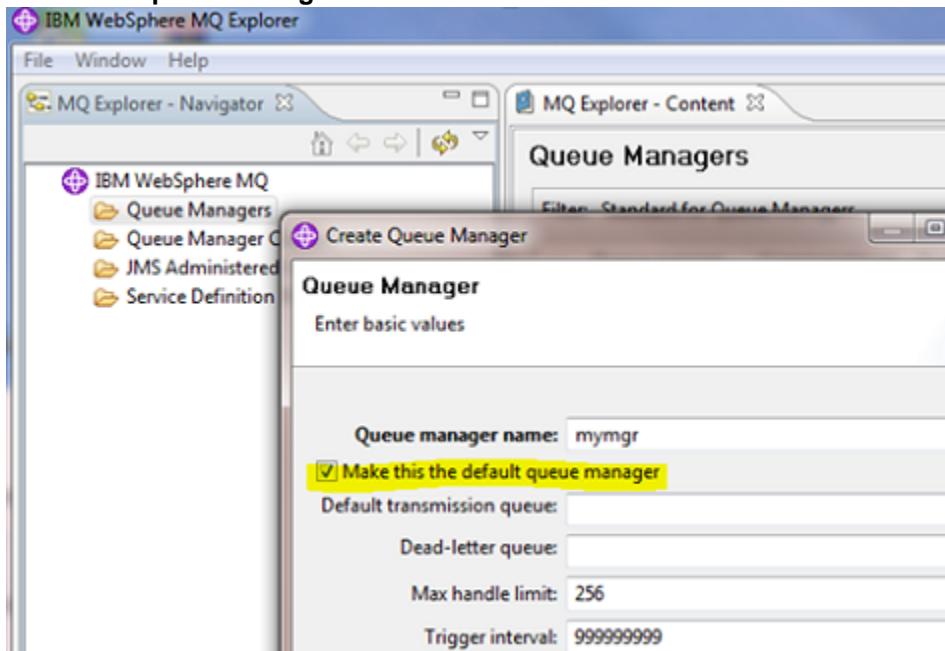
# appropriate one should be uncommented.
#
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
#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
.....

```

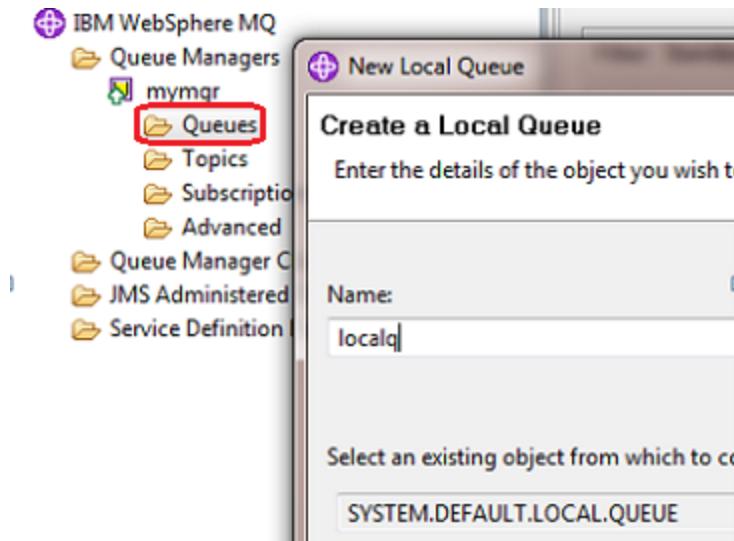
6. Restart the WebSphere MQ service.

Creating the Queue in WebSphere MQ

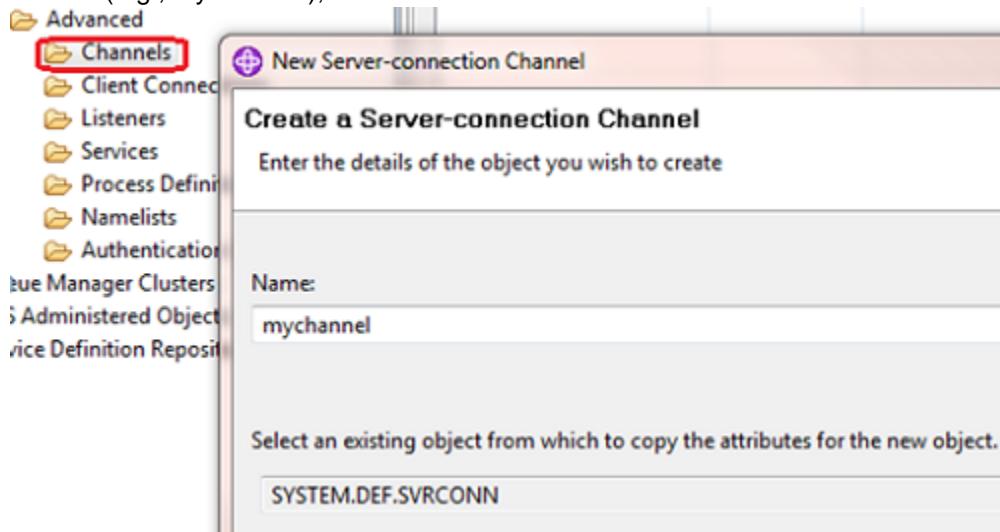
1. Start IBM WebSphere MQ Explorer, and create a new queue manager. Make sure you select the **make this the default queue manager** check box and leave the default values on the other fields as shown below.



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.

```
IVTRun.bat -url "file:/C:/JNDI-Directory" -icf
com.sun.jndi.fscontext.RefFSContextFactory
```

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:

```
ALTER TCF(ivtTCF) TRANSPORT(CLIENT)
ALTER TCF(ivtTCF) QMGR(mymgr)
```

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 DAS

Follow the instructions below to configure WSO2 DAS.

1. If you set up WSO2 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 <DAS_HOME>/repository/components/lib/ directory .
 - com.ibm.mqjms.jar
 - fscontext.jar
 - providerutil.jar
 - com.ibm.mq.jmqi.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.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 <DAS_HOME>/repository/components/dropins/ directory.
 - com.ibm.mq_2.0.0.jar
 - fscontext_1.0.0.jar
 5. Add the following files to the <DAS_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 DAS (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 `<Das_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</pa
rameter>
                    <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"
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">queue</parameter>
                                </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*	IBMWebSphereJMSInputEventAdapter ⑦ Enter a unique name to identify Event Receiver
From	jms ⑦ Select the type of Adapter to receive events
Adapter Properties	
Topic/Queue Name*	test_topic ⑦ Topic/Queue name of the input stream
JNDI Initial Context Factory Class*	com.sun.jndi.fscontext.RefFSContextFactory ⑦ JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.
JNDI Provider URL*	file:/C:/JNDI-Directory ⑦ URL of the JNDI provider.
The JMS connection password	
The JMS connection username	
Connection Factory JNDI Name*	lv1QCF ⑦ The JNDI name of the connection factory.
Destination Type*	topic ⑦ Type of the destination.
Enable Durable Subscription	false ⑦ Whether the subscription is durable or not.
Durable Subscriber Name	⑦ Name of the durable subscriber (If any value added, Durable subscription will be enabled).
JMS Properties	⑦ Axis2 JMS Properties, e.g. "property1: value1, property2: value2"
To	
Event Stream*	Test Stream:1.0.0 ⑦ The event stream that will be generated by the received events
Mapping Configuration	
Message Format*	xml ⑦ Select the input message format
+ Advanced	
<input type="button" value="Add 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="iBMWebSphereJMSInputEventAdapter"
    statistics="disable" trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
    <from eventAdapterType="jms">
        <property
name="java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFactory</propert
y>
            <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.ConnectionFactoryJNDIName">ivtQCF</property>
            <property encrypted="true"
name="transport.jms.Password">P2ve4G8+qF7JXkiGnP9/Ew5GXALEWTu7znEwxZGYa/MQMaQBRfsXiP09
4fn9U+0rntdBMitXU9o7h5uV3m5h97Po8WTJRpnFBV5YCGZEO+ELSG6twY3386MipwFhFMrbUMKamI2sXksDRc
ogojWKtoHNmODnt8Ud1dh0LK5zqec=          </property>
            <property name="transport.jms.UserName">jms-user</property>
            <property name="transport.jms.DurableSubscriberClientID">subscriber</property>
            <property name="jms.properties">SessionTransacted:false</property>
        </from>
        .....
    </eventReceiver>

```

The above adapter properties are described below.

Adapter Property	Description	Configuration file property
Topic/Queue Name	A valid name for the JMS topic/queue. WSO2 CE P/DAS sends and receives messages by subscribing to a topic or using named queues.	transport.jms.Destination
JNDI Initial Context Factory Class	JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.	java.naming.factory.initial
J N D I Provider URL	URL of the JNDI provider.	java.naming.provider.url
The JMS connection password	A valid password for the JMS connection	transport.jms.Password
The JMS connection username	A valid username for the JMS connection	transport.jms.UserName
Connection Factory JNDI Name	The JNDI name of the connection factory.	transport.jms.ConnectionFactoryJNDI
Destination Type	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 Client ID	A string of characters to denote a valid id for client connecting as the durable subscriber. (It will enable durable subscription if you add any value here).	transport.jms.DurableSubscriberClientID
JMS Properties	<p>Valid "property:value" pairs of Axis2 JMS properties (e.g. "property1: value1, property2: value2")</p> <p>For more information on Axis2 JMS properties, go to Apache AXIS2 Transports Documentation.</p>	jms.properties

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. Add the following broker-specific JAR files to <PRODUCT_HOME>/repository/components/lib/ directory.
 - <ACTIVEMQ_HOME>/lib/geronimo-jms_1.1_spec-1.0.jar
 - <QPID-CLIENT_HOME>/lib/ qpid-client-xxx.jar
 - <QPID-CLIENT_HOME>/lib/ qpid-common-xxx.jar
3. 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

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="QpidJMSInputEventAdapter"/> <small>⑦ Enter a unique name to identify Event Receiver</small>	From <input type="text" value="jms"/> <small>⑦ Select the type of Adapter to receive events</small>
Adapter Properties	
Topic/Queue Name* <input type="text" value="test-topic"/> <small>⑦ Topic/Queue name of the input stream</small>	JNDI Initial Context Factory Class* <input type="text" value="arg.apache.qpid.jndi.PropertiesFileInitialContextFactory"/> <small>⑦ JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</small>
JNDI Provider URL* <input type="text" value="repository/conf/jndi.properties"/> <small>⑦ URL of the JNDI provider.</small>	
The JMS connection password <input type="text"/>	
The JMS connection username <input type="text"/>	
Connection Factory JNDI Name* <input type="text" value="TopicConnectionFactory"/> <small>⑦ The JNDI name of the connection factory.</small>	
Destination Type* <input type="text" value="topic"/> <small>⑦ Type of the destination.</small>	
Enable Durable Subscription <input type="text" value="false"/> <small>⑦ Whether the subscription is durable or not.</small>	
Durable Subscriber Client ID <input type="text"/> <small>⑦ Name/ID of the durable subscriber (If any value added, Durable subscription will be enabled).</small>	
JMS Properties <input type="text"/> <small>⑦ Axis2 JMS Properties, e.g. "property1: value1, property2: value2"</small>	
To	
Event Stream* <input type="text" value="TestStream:1.0.0"/> <small>⑦ The event stream that will be generated by the received events</small>	
Mapping Configuration	
Message Format* <input type="text" value="xml"/> <small>⑦ Select the input message format</small>	
 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="QpidJMSInputEventAdapter" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventreceiver">
    <from eventAdapterType="jms">
        <property
name="java.naming.factory.initial">org.apache.qpid.jndi.PropertiesFileInitialContextFa
ctory</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/Ew5GXALEWTu7znEwxZGYa/MQMaQBRfsXiP09
4fn9U+0rntdBMitXU9o7h5uV3m5h97Po8WTJRpnFBV5YCGZEO+ELSz6twY3386MipwFhFMrbUMKamI2sXksDRc
ogojWKtoHNmODnt8Ud1dh0LK5zqec=      </property>
        <property name="transport.jms.UserName">jms-user</property>
        <property name="transport.jms.DurableSubscriberClientID">subscriber</property>
        <property name="jms.properties">SessionTransacted:false</property>
    </from>
    .....
</eventReceiver>

```

The above adapter properties are described below.

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. 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 an TIBCO JMS event receiver using the management console as shown below.

Event Receiver Details

Enter Event Receiver Details	
Event Receiver Name*	TibcoJmsReceiver
From	
Input Event Adapter Type*	jms ▾
Adapter Properties	
Topic/Queue Name *	Test Topic ② Topic/Queue name of the input stream
JNDI Initial Context Factory Class *	com.tibco.tibjms.naming.TibjmsInitialContextFactory ② JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.
JNDI Provider URL *	tibjmsnaming://localhost:7222 ② URL of the JNDI provider.
The JMS connection password	*****
The JMS connection username	jms-user
Connection Factory JNDI Name *	TopicConnectionFactory ② The JNDI name of the connection factory.
Destination Type *	topic ② Type of the destination.
Enable Durable Subscription	true ② Whether the subscription is durable or not.
Durable Subscriber Name	subscriber ② Name of the durable subscriber (If any value added, Durable subscription will be enabled).
JMS Properties	② Axis2 JMS Properties, e.g. "property1: value1, property2: value2"
To	
Event Stream*	testJMSStream:1.0.0 ▾
Mapping Configuration	
Message Format*	xml ▾

When configuring WSO2 DAS for fail over connections, specify JNDI Provider URLs as a comma-separated list of URLs.
e.g: tibjmsnaming://localhost:7222, tibjmsnaming://localhost:7224

WSO2 DAS will attempt to connect to each URL in the ordered list. If a connection to one URL fails, the WSO2 DAS 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** that you want to apply to 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="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">JP4yDiEh6HogOEjJzQQwHaJFIWZlnJTzaERl4eYrwukNeypm36R+odMk
aN9b2q4H9jbQsRV+mhcT1wQVnBpEZn4a+SuFuLKh3NihDEgww6R1tZVo8p1D6TUKvSHXYEpwS0gKrk0mdaFEOQ
0jfdhfK3Hrnjkz/MYPYQknrLK5MIY=</property>
                <property name="transport.jms.UserName">jms-user</property>
            </from>
            .....
    </eventReceiver>

```

The above adapter properties are described below.

Adapter Property	Description	Configuration file property
Topic/Queue Name	A string of characters to denote a valid name of a JMS topic to subscribe to, or named queue to use when WSO2 DAS sends and receives messages.	transport.jms.Destination
JNDI Initial Context Factory Class	JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.	java.naming.factory.initial
J N D I Provider URL	URL of the JNDI provider.	java.naming.provider.url
The JMS connection password	A valid password for the JMS connection.	transport.jms.Password
The JMS connection username	A valid username for the JMS connection.	transport.jms.UserName
Connection Factory JNDI Name	The JNDI name of the connection factory.	transport.jms.ConnectionFactoryJNDI:

Destination Type	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 string of characters to denote a valid name of the durable subscriber. (It enables durable subscription if you add any value here).	transport.jms.DurableSubscriberName
J M S Properties	<p>Valid property and value pairs to denote Axis2 JMS properties (e.g. "property1: value1, property2: value2")</p> <p>For more information on Axis2 JMS properties, go to Apache AXIS2 Transports Documentation.</p>	jms.properties

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](#).
2. Add the following JMS -specific JAR files to <PRODUCT_HOME>/repository/components/lib/ directory
 - <WSO2MB_HOME>/client-lib/andes-client-xx.jar
 - <WSO2MB_HOME>/client-lib/geronimo-j2ee-management_1.1_spec-1.0.1xx.jar

Creating a WSO2 MB JMS event receiver

For instructions on creating a WSO2 MB JMS event receiver, see [Configuring Event Receivers](#).

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

Event Receiver Name* <input type="text" value="WSO2MBJMSInputEventAdapter"/> <small>⑦ Enter a unique name to identify Event Receiver</small>	From Input Event Adapter Type* <input type="text" value="jms"/> <small>⑦ Select the type of Adapter to receive events</small>
Adapter Properties	
Topic/Queue Name* <input type="text" value="test_topic"/> <small>⑦ Topic/Queue name of the input stream</small>	JNDI Initial Context Factory Class* <input type="text" value="org.wso2.andes.jndi.PropertiesFileInitialContextFactory"/> <small>⑦ JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</small>
JNDI Provider URL* <input type="text" value="repository/conf/jndi.properties"/> <small>⑦ URL of the JNDI provider.</small>	The JMS connection password <input type="text"/>
The JMS connection username <input type="text"/>	Connection Factory JNDI Name* <input type="text" value="TopicConnectionFactory"/> <small>⑦ The JNDI name of the connection factory.</small>
Destination Type* <input type="text" value="topic"/> <small>⑦ Type of the destination.</small>	Enable Durable Subscription <input type="text" value="false"/> <small>⑦ Whether the subscription is durable or not.</small>
Durable Subscriber Client ID <input type="text"/> <small>⑦ Name/ID of the durable subscriber (If any value added, Durable subscription will be enabled).</small>	
JMS Properties <input type="text"/> <small>⑦ Axis2 JMS Properties, e.g. "property1: value1, property2: value2"</small>	
To	
Event Stream* <input type="text" value="TestStream:1.0.0"/> <small>⑦ The event stream that will be generated by the received events</small>	
Mapping Configuration	
Message Format* <input type="text" value="xml"/> <small>⑦ Select the input message format</small>	
+ 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="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/Ew5GXALEWTu7znEwxZGYa/MQMaQBRfsXiP09
4fn9U+0rntdBMitXU9o7h5uV3m5h97Po8WTJRpnFBV5YCGZEO+ELSg6twY3386MipwFhFMrbUMKamI2sXksDRc
ogojWKtoHNmODnt8Ud1dh0LK5zqec=      </property>
        <property name="transport.jms.UserName">jms-user</property>
        <property name="transport.jms.DurableSubscriberClientID">subscriber</property>
        <property name="jms.properties">SessionTransacted:false</property>
    </from>
    .....
</eventReceiver>

```

The above adapter properties are described below.

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. Copy the following client JAR files from <KAFKA_HOME>/lib/ directory to <PRODUCT_HOME>/repository /components/lib/ directory.
 - kafka_2.10-0.8.1.jar
 - zkclient-0.3.jar
 - scala-library-2.10.1.jar
 - zookeeper-3.3.4.jar
 - kafka-clients-0.8.2.1
 - metrics-core-2.2.0

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.

If you are using Kafka_2.11-0.10.0.0, you need to download [this jar](#) and save it in the <PRODUCT_HOME>/repository/components/lib/ directory.

3. Start the Apache Kafka server.

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](#)

Enter Event Receiver Details

Event Receiver Name*	KafkaInputEventAdapter <small>Enter a unique name to identify Event Receiver</small>
From	
Input Event Adapter Type*	kafka <small>Select the type of Adapter to receive events</small>
Adapter Properties	
Server Zookeeper IP*	127.0.0.1 <small>IP address of the Zookeeper Server (eg: 127.0.0.1)</small>
Group ID Kafka*	groupid <small>Kafka consumer group id</small>
Threads*	4 <small>No of consumer threads</small>
Optional Configuration Properties	
Topic Kafka*	test_topic
Is events duplicated in cluster	false
To	
Event Stream*	Test Stream:1.0.0 <small>The event stream that will be generated by the received events</small>
Mapping Configuration	
Message Format*	xml <small>Select the input message format</small>
Advanced	
<input type="button" value="Add 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** that 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="KafkaInputEventAdapter" 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.

Adapter Property	Description	Configuration file property	Example
Server Zookeeper IP	IP address of the Zookeeper Server	zookeeper.connect	127.0.0.1
Group Kafka ID	Kafka consumer group id which uniquely identifies a set of consumers within the same consumer group	group.id	groupid
Threads	Number of consumer threads	threads	4
Optional Configuration Properties	Valid property and value pairs to denote optional configuration properties for Apache Kafka. (E.g. "property1: value1, property2: value2") For more information on Axis2 JMS properties, go to Apache Kafka Documentation .	optional.configuration	zk.sessiontimeout.ms:6000
Topic Kafka	Name of the Kafka topic to which, input messages are published	topic	test_topic
Is events duplicated in cluster	In a cluster whether the same event can reach two receiver nodes	events.duplicated.in.cluster	true/false

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 [Java](#).

SON, 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. Download **MQTT client library** (`mqtt-client-0.4.0.jar`).
2. Add the JAR file to the `<PRODUCT_HOME>/repository/components/lib/` directory.
3. 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.

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name* MQTTInputEventAdapter
Enter a unique name to identify Event Receiver

From

Input Event Adapter Type* mqtt
Select the type of Adapter to receive events

Adapter Properties

Topic*	<input type="text" value="test_topic"/> <small>test_topic Topic subscribed</small>
Broker Url*	<input type="text" value="tcp://localhost:1883"/> <small>tcp://localhost:1883 MQTT broker url</small>
Username	<input type="text"/>
Password	<input type="text"/>
Clean Session	<input type="text" value="true"/> <small>true Persist topic subscriptions and ack positions across client sessions</small>
Client Id	<input type="text"/> <small>client identifier is used by the server to identify a client when it reconnects, It used for durable subscriptions or reliable delivery of messages is required.</small>

To

Event Stream* Test Stream:1.0.0
The event stream that will be generated by the received events

Mapping Configuration

Message Format* text
Select the input message format

[Advanced](#)

[Add 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** that 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="MQTTInputEventAdapter" 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
Client Id	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.

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="SOAPReceiver"/> ? Enter a unique name to identify Event Receiver	
From	
Input Event Adapter Type* <input type="text" value="soap"/> ? Select the type of Adapter to receive events	
<p>Following url formats are used to receive events For super tenants: <code>http://localhost:9764/services/<event_receiver_name>/receive</code> <code>https://localhost:9444/services/<event_receiver_name>/receive</code> <code>local:///services/<event_receiver_name>/receive</code></p>	
Usage Tips <p>For other tenants: <code>http://localhost:9764/services/t/<tenant_domain>/<event_receiver_name>/receive</code> <code>https://localhost:9444/services/t/<tenant_domain>/<event_receiver_name>/receive</code> <code>local:///services/t/<tenant_domain>/<event_receiver_name>/receive</code></p>	
Adapter Properties	
Transport(s)* <input type="text" value="http"/> ? 	
To	
Event Stream* <input type="text" value="TestStream:1.0.0"/> ? The event stream that will be generated by the received events	
Mapping Configuration	
Message Format* <input type="text" value="xml"/> ? Select the input message format	
+ Advanced	
<input type="button" value="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** that 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="SOAPReceiver" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventreceiver">
    <from eventAdapterType="soap">
        <property name="transports">http</property>
    </from>
    .....
</eventReceiver>

```

The above adapter properties are described below.

Adapter property	Description	Configuration file property	Example
Transport(s)	Transport type via which the events are received.	transports	http

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.

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="WebSocketInputEventAdapter"/> <div style="font-size: small; color: #808080;">? Enter a unique name to identify Event Receiver</div>	From <input type="text" value="websocket"/> <div style="font-size: small; color: #808080;">? Select the type of Adapter to receive events</div>
Adapter Properties	
Web Socket Server URL* <input type="text" value="websocket.server.url"/> <div style="font-size: small; color: #808080;">? URL of the web socket server you want to connect to, e.g. "ws://localhost:9099".</div>	Is events duplicated in cluster <input type="text" value="false"/>
To	
Event Stream* <input type="text" value="TestStream:1.0.0"/> <div style="font-size: small; color: #808080;">? The event stream that will be generated by the received events</div>	
Mapping Configuration	
Message Format* <input type="text" value="xml"/> <div style="font-size: small; color: #808080;">? Select the input message format</div>	+ Advanced
<input type="button" value="Add 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** that 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="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.

Adapter Property	Description	Configuration file property	Example
Web Socket Server URL	URL of the WebSocket server to which you want to connect to	websocket.server.url	ws://localhost:9099
Is events duplicated in cluster	In a cluster whether the same event can reach two receiver nodes	events.duplicated.in.cluster	true/false

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 DAS for the DAS to receive events. However, if the event source is a web socket server, the DAS 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.

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="WebSocketLocalInputEventReceiver"/> <div style="font-size: small; margin-top: -10px;">⑦ Enter a unique name to identify Event Receiver</div>	
From	
Input Event Adapter Type* <input type="text" value="websocket-local"/> <div style="font-size: small; margin-top: -10px;">⑦ Select the type of Adapter to receive events</div>	
Usage Tips <div style="margin-left: 20px;">Following url formats are used to receive events: <i>ws://localhost:9764/inputwebsocket/<receiver_name></i> <i>wss://localhost:9444/inputwebsocket/<receiver_name></i></div>	
To	
Event Stream* <input type="text" value="TestStream:1.0.0"/> <div style="font-size: small; margin-top: -10px;">⑦ The event stream that will be generated by the received events</div>	
Mapping Configuration	
Message Format* <input type="text" value="xml"/> <div style="font-size: small; margin-top: -10px;">⑦ Select the input message format</div>	
+ Advanced	

[Add Event Receiver](#)

When multi-tenancy is used, the URL formats used to receive events are as follows.

Super-tenant/Tenant	URL Formats
Super-tenant	<pre>ws://localhost:";<DAS_Server_Port>"/inputwebsocket/<receiver_name> wss://localhost:";<DAS_SSL_Server_Port>"/inputwebsocket/<receiver_name></pre>
Tenant	<pre>ws://localhost:";<DAS_Server_Port>"/t/<tenant_domain>/inputwebsocket/<receiver_name> wss://localhost:";<DAS_SSL_Server_Port>"/t/<tenant_domain>/inputwebsocket/<receiver_name>"</pre>

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.

Super-tenant/Tenant	URL Formats
Super-tenant	<pre>ws://localhost:";9763"/inputwebsocket/WebSocketLocalInputEventReceiver wss://localhost:";9443"/inputwebsocket/WebSocketLocalInputEventReceiver</pre>

Tenant	<pre>ws://localhost:";9763"/t/mycompany.com/inputwebsocket/WebSocketEventReceiver wss://localhost:";9443"/t/mycompany.com/inputwebsocket/WebSocketEventReceiver";</pre>
--------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="WSO2EventReceiver"/> <div style="font-size: small; margin-top: -10px;">② Enter a unique name to identify Event Receiver</div> From Input Event Adapter Type* <input type="text" value="wso2event"/> <div style="font-size: small; margin-top: -10px;">② Select the type of Adapter to receive events</div> <div style="margin-top: 10px;"> Following url formats are used to receive events For load-balancing: use ";" to separate values for multiple endpoints Eg: {tcp://<hostname>:<port>},tcp://<hostname>:<port>, ...} </div> <div style="margin-top: 10px;"> For failover: use " " to separate multiple endpoints Eg: {tcp://<hostname>:<port>}/{tcp://<hostname>:<port>} ... </div> <div style="margin-top: 10px;"> For more than one cluster: use "[" to separate multiple clusters Eg: {tcp://<hostname>:<port>}/{tcp://<hostname>:<port>}/ ...},{tcp://<hostname>:<port>} </div> <div style="margin-top: 10px;"> Ports available for Thrift protocol – TCP port:7612 or SSL port:7712 Ports available for Binary protocol – TCP port:9612 or SSL port:9712 </div> Usage Tips	Adapter Properties <div style="margin-bottom: 10px;"> <input type="text" value="false"/> <div style="font-size: small; margin-top: -10px;">② This depends on how events are published to the server, 'true' only if multiple receiver URLs are defined in different receiver groups ([]).</div> </div> To Event Stream* <input type="text" value="TestStream:1.0.0"/> <div style="font-size: small; margin-top: -10px;">② The event stream that will be generated by the received events</div> Mapping Configuration Message Format* <input type="text" value="wso2event"/> <div style="font-size: small; margin-top: -10px;">② Select the input message format</div> <div style="margin-top: 10px;"> + Advanced </div>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

[Add 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** that 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.

Adapter Property	Description	Configuration file property	Example
Is events duplicated in cluster	In a cluster whether the same event can reach two receiver nodes	events.duplicated.in.cluster	true/false

Related samples

For more information on wso2event event receiver type, see the following sample.

- 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

Message Format* wso2event   Select the input message format

 Advanced

WSO2Event Mapping

Input Event Stream *

Input Event Version *

Meta Data

Input Attribute Name :	<input type="text" value="time"/>	Mapped To :	<input type="text" value="meta_timestamp"/>	Attribute Type :	<input type="text" value="long"/>
------------------------	-----------------------------------	-------------	---------------------------------------------	------------------	-----------------------------------

Input Attribute Name :	<input type="text" value="meta_ispowerServed"/>	Mapped To :	<input type="text" value="meta_isPowerSaverEnable"/>	Attribute Type :	<input type="text" value="bool"/>
------------------------	-------------------------------------------------	-------------	------------------------------------------------------	------------------	-----------------------------------

Input Attribute Name :	<input type="text" value="id"/>	Mapped To :	<input type="text" value="meta_sensorId"/>	Attribute Type :	<input type="text" value="int"/>
------------------------	---------------------------------	-------------	--------------------------------------------	------------------	----------------------------------

Input Attribute Name :	<input type="text" value="name"/>	Mapped To :	<input type="text" value="meta_sensorName"/>	Attribute Type :	<input type="text" value="string"/>
------------------------	-----------------------------------	-------------	----------------------------------------------	------------------	-------------------------------------

Correlation Data

Input Attribute Name :	<input type="text" value="correlation_longitude"/>	Mapped To :	<input type="text" value="correlation_longitude"/>	Attribute Type :	<input type="text" value="double"/>
------------------------	----------------------------------------------------	-------------	----------------------------------------------------	------------------	-------------------------------------

Input Attribute Name :	<input type="text" value="correlation_latitude"/>	Mapped To :	<input type="text" value="correlation_latitude"/>	Attribute Type :	<input type="text" value="double"/>
------------------------	---------------------------------------------------	-------------	---------------------------------------------------	------------------	-------------------------------------

Payload Data

Input Attribute Name :	<input type="text" value="humid"/>	Mapped To :	<input type="text" value="humidity"/>	Attribute Type :	<input type="text" value="float"/>
------------------------	------------------------------------	-------------	---------------------------------------	------------------	------------------------------------

Input Attribute Name :	<input type="text" value="value"/>	Mapped To :	<input type="text" value="sensorValue"/>	Attribute Type :	<input type="text" value="double"/>
------------------------	------------------------------------	-------------	------------------------------------------	------------------	-------------------------------------

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.

```
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.

Mapping Configuration

Message Format* xml Select the input message format

+ Advanced

XML Mapping

XPath Prefixes

Prefix	Namespace	Actions
sen	http://wso2.org	Delete

Prefix : Namespace : Add

Parent Selector XPath Expression:

Properties

XPath:	<input type="text" value="//sen:time"/>	Mapped To:	<input type="text" value="meta_timestamp"/>	Type:	<input type="text" value="long"/>	Default Value:	<input type="text"/>
XPath:	<input type="text" value="//sen:isPowerSaverEnabled"/>	Mapped To:	<input type="text" value="meta_isPowerSaverEnabled"/>	Type:	<input type="text" value="bool"/>	Default Value:	<input type="text" value="true"/>
XPath:	<input type="text" value="//sen:id"/>	Mapped To:	<input type="text" value="meta_sensorId"/>	Type:	<input type="text" value="int"/>	Default Value:	<input type="text"/>
XPath:	<input type="text" value="//sen:name"/>	Mapped To:	<input type="text" value="meta_sensorName"/>	Type:	<input type="text" value="string"/>	Default Value:	<input type="text"/>
XPath:	<input type="text" value="//sen:long"/>	Mapped To:	<input type="text" value="correlation_longitude"/>	Type:	<input type="text" value="double"/>	Default Value:	<input type="text"/>
XPath:	<input type="text" value="//sen:lat"/>	Mapped To:	<input type="text" value="correlation_latitude"/>	Type:	<input type="text" value="double"/>	Default Value:	<input type="text"/>
XPath:	<input type="text" value="//sen:humidity"/>	Mapped To:	<input type="text" value="humidity"/>	Type:	<input type="text" value="float"/>	Default Value:	<input type="text"/>
XPath:	<input type="text" value="//sen:value"/>	Mapped To:	<input type="text" value="sensorValue"/>	Type:	<input type="text" value="double"/>	Default Value:	<input type="text"/>

The configuration XML file of the above sample mapping is as follows.

```

<eventReceiver ... xmlns="http://wso2.org/carbon/eventreceiver">
    <from ... />
    <mapping customMapping="enable" type="xml">
        <xpathDefinition namespace="http://wso2.org" prefix="sen"/>
        <property>
            <from xpath="//sen:time" />
            <to name="meta_timestamp" type="long"/>
        </property>
        <property>
            <from xpath="//sen:isPowerSeved" />
            <to default="true" name="meta_isPowerSaverEnabled" type="bool"/>
        </property>
        <property>
            <from xpath="//sen:id" />
            <to name="meta_sensorId" type="int"/>
        </property>
        <property>
            <from xpath="//sen:name" />
            <to name="meta_sensorName" type="string"/>
        </property>
        <property>
            <from xpath="//sen:long" />
            <to name="correlation_longitude" type="double"/>
        </property>
        <property>
            <from xpath="//sen:lat" />
            <to name="correlation_latitude" type="double"/>
        </property>
        <property>
            <from xpath="//sen:humidity" />
            <to name="humidity" type="float"/>
        </property>
        <property>
            <from xpath="//sen: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 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
- **String Attributes**
 - null - It will be processed as a null value
 - Empty Attribute - It will be processed as an empty string

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 screenshot shows the 'Mapping Configuration' interface. At the top, there is a dropdown menu labeled 'json' with a tooltip 'Select the input message format'. Below it is a 'Advanced' button. The main area is titled 'JSON Mapping' and contains a table of 'Available JSON Mappings'. The table has columns for 'JSONPath', 'Mapped To', 'Type', and 'Default Value'. The mappings listed are:

JSONPath	Mapped To	Type	Default Value
\$.sensorData.time	meta_timestamp	long	
\$.sensorData.powerSaverEnabled	meta_isPowerSaverEnabled	bool	true
\$.sensorData.id	meta_sensorId	int	
\$.sensorData.name	meta_sensorName	string	---
\$.sensorData.long	correlation_longitude	double	
\$.sensorData.lat	correlation_latitude	double	
\$.sensorData.humidity	humidity	float	
\$.sensorData.value	sensorValue	double	

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**
 - null - It will be processed as a null value
 - Empty Attribute - It will be processed as an empty string

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 screenshot shows the 'Mapping Configuration' screen with the 'Text Mapping' tab selected. At the top, there is a dropdown menu labeled 'text' with a tooltip 'Select the input message format'. Below it is an 'Advanced' button. The main area is divided into two sections: 'Text Mapping' and 'Available Text Mappings'.

Text Mapping:

- Regular Expression Definitions:** A table listing four regular expression definitions:

Regular Expression	Actions
(\w+)\s(\w+)\s(\w+)\s(\w+)	Delete
time\s(\w+)	Delete
powerSaved\s(\w+)	Delete
value\s(\w+)\s(\w+)	Delete
- Regular Expression:** An input field with a placeholder 'Regular Expression : *' and an 'Add' button.

Available Text Mappings:

Regular Expression	Mapped To	Type	Default Value	Actions
(\w+)\s(\w+)\s(\w+)\s(\w+)	meta_sensorid	int		Delete
(\w+)\s(\w+)\s(\w+)\s(\w+)	meta_sensorName	string	--	Delete
(\w+)\s(\w+)\s(\w+)\s(\w+)	correlation_longitude	double		Delete
(\w+)\s(\w+)\s(\w+)\s(\w+)	correlation_latitude	double		Delete
time\s(\w+)	meta_timestamp	long		Delete
powerSaved\s(\w+)	meta_isPowerSaverEnabled	bool		Delete
value\s(\w+)\s(\w+)	humidity	float		Delete
value\s(\w+)\s(\w+)	sensorValue	double		Delete

Below the table are fields for defining a new mapping: 'Regular Expression : value\s(\w+)\s(\w+)', 'Mapped To : sensorValue', 'Type: double', 'Default Value : ', and an 'Add' button.

The configuration XML file of the above sample mapping is as follows.

```

<eventReceiver ... xmlns="http://wso2.org/carbon/eventreceiver">
    <from ... />
    <mapping customMapping="enable" type="text">
        <property>
            <from regex="(\w+)\s(\w+)\s(\w+)\s(\w+)" />
            <to name="meta_sensorId" type="int" />
            <to default="--" name="meta_sensorName" type="string" />
            <to name="correlation_longitude" type="double" />
            <to name="correlation_latitude" type="double" />
        </property>
        <property>
            <from regex="time\s(\w+)" />
            <to name="meta_timestamp" type="long" />
        </property>
        <property>
            <from regex="powerSaved\s(\w+)" />
            <to name="meta_isPowerSaverEnabled" type="bool" />
        </property>
        <property>
            <from regex="value\s(\w+)\s(\w+)" />
            <to name="humidity" type="float" />
            <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 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.

Mapping Configuration

Message Format*  Select the input message format

 Advanced

Map(Key/Value) Mapping**Available Map(Key/Value) Mappings**

Input Attribute Name :	<input type="text" value="timestamp"/>	Mapped To :	<input type="text" value="meta_timestamp"/>	Type: long
Input Attribute Name :	<input type="text" value="isPowerSaverEnabled"/>	Mapped To :	<input type="text" value="meta_isPowerSaverEnabled"/>	Type: bool
Input Attribute Name :	<input type="text" value="id"/>	Mapped To :	<input type="text" value="meta_sensorId"/>	Type: int
Input Attribute Name :	<input type="text" value="name"/>	Mapped To :	<input type="text" value="meta_sensorName"/>	Type: string
Input Attribute Name :	<input type="text" value="long"/>	Mapped To :	<input type="text" value="correlation_longitude"/>	Type: double
Input Attribute Name :	<input type="text" value="lat"/>	Mapped To :	<input type="text" value="correlation_latitude"/>	Type: double
Input Attribute Name :	<input type="text" value="humidity"/>	Mapped To :	<input type="text" value="humidity"/>	Type: float
Input Attribute Name :	<input type="text" value="sensorValue"/>	Mapped To :	<input type="text" value="sensorValue"/>	Type: double

The configuration XML file of the above sample mapping is as follows.

```

<eventReceiver ... xmlns="http://wso2.org/carbon/eventreceiver">
    <from ... />
    <mapping customMapping="enable" type="map">
        <property>
            <from name="timestamp" />
            <to name="meta_timestamp" type="long" />
        </property>
        <property>
            <from name="isPowerSaverEnabled" />
            <to name="meta_isPowerSaverEnabled" type="bool" />
        </property>
        <property>
            <from name="id" />
            <to name="meta_sensorId" type="int" />
        </property>
        <property>
            <from name="name" />
            <to name="meta_sensorName" type="string" />
        </property>
        <property>
            <from name="long" />
            <to name="correlation_longitude" type="double" />
        </property>
        <property>
            <from name="lat" />
            <to name="correlation_latitude" type="double" />
        </property>
        <property>
            <from name="humidity" />
            <to name="humidity" type="float" />
        </property>
        <property>
            <from name="sensorValue" />
            <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 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
 - Empty Attribute - It will be processed as an empty string

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.adapter.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 eventAdapterListener) 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:

```
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_DUPLICATED_IN_C
LUSTER));
    }

    @Override
    public boolean isPolling() {
        return true;
    }

    private void validateInputEventAdapterConfigurations() throws
InputEventAdapterException {
        String delayInMillisProperty =
eventAdapterConfiguration.getProperties().get(FileTailEventAdapterConstants.EVENT_ADAP
TER_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_ADAP
TER_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_ADAP

```

```
TER_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);
```

```

        singleThreadedExecutor.execute(tailer);
    }
}

```

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:

```

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(
resourceBundle.getString(FileTailEventAdapterConstants.EVENT_ADAPTER_CONF_FILEPATH));
        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_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 “**“InputEventAdapterFactory”** type. Below is a sample implementation for a service class for a File Tail Receiver:

```

/**
 * @scr.component component.name="input.File.AdapterService.component"
immediate="true"
 * @scr.reference name="configurationcontext.service"
 * interface="org.wso2.carbon.utils.ConfigurationContextService" cardinality="1..1"
 * policy="dynamic" bind="setConfigurationContextService"
unbind="unsetConfigurationContextService"
 */

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(configurationCon
textService);
    }

    protected void unsetConfigurationContextService(
        ConfigurationContextService configurationContextService) {

        FileTailEventAdapterServiceHolder.unregisterConfigurationContextService(configurationC
ontextService);
    }
}
```

Furthermore you can have a utility directory as ***internal\util*** where you can place utility classes required for the custom receiver implementation.

Deploying Custom Event Receiver

Deploying a custom event receiver is very simple in WSO2 DAS 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 **<Das_Home>/repository/components/dropins**. In DAS server startup, you can see the newly created event receiver type service in 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.

Configuring Received Data

This section describes the additional configurations that can be done to data collected by DAS before they are processed.

Setting a custom timestamp value for incoming events

When an event is received by WSO2 DAS, the timestamp of that event is set as a parameter value. This value often needs to be overridden in many situations including the following examples.

- Incremental data processing
- Timestamp based Analytics

In order to override the timestamp value assigned at the time an event is received, the client that publishes events to WSO2 DAS should set an attribute named `_timestamp` of the `Long` type as a payload attribute in each event.

Querying for the timestamp in a Spark script

You can query for events by the timestamp in an Analytics script. In order to do this, you need to add a `_timestamp` option in the table schema.

e.g., If there is a table named `TestTable1` in WSO2 DAS with the name `STRING`, `b INT` schema, the timestamps of the events in the table can be queried as follows.

```
CREATE TEMPORARY TABLE table1 USING CarbonAnalytics OPTIONS (tableName "TestTable1",
schema "a1 STRING, b1 INT, _timestamp LONG") ;
SELECT a1, _timestamp FROM table1;
```

Additionally, you can use the `_timestamp` option to insert a custom value for the timestamp parameter of an event.

e.g., If you need to insert a set of values from the `table1` table to another table named `table2`, you can write a query as follows.

```
CREATE TEMPORARY TABLE table2 USING CarbonAnalytics OPTIONS (tableName "TestTable2",
schema "a2 STRING, _timestamp LONG") ;
```

The following query inserts the results of the `SELECT a1, _timestamp FROM table1` query into the `TestTable2` table.

```
INSERT INTO TABLE table2 SELECT a1, _timestamp FROM table1;
```

You can override the behavior of the `_timestamp` field to use the current timestamp of the event, by setting `-1` as shown below.

```
INSERT INTO TABLE table2 SELECT a1, -1 as _timestamp FROM table1;
```

Analyzing Data

After collecting and storing data, the next step in analyzing data is to analyze them to produce meaningful information. WSO2 DAS's analyzer engine retrieves data from the data stores and performs various analysis operations on them according to defined analytic queries.

The following sections describe how to work with these components to analyze your stored data:

- [Realtime Analytics Using Siddhi](#)
- [Interactive Analytics](#)
- [Batch Analytics Using Spark SQL](#)
- [Predictive Analytics](#)

Realtime Analytics Using Siddhi

Following sections describe how you can perform real-time analytics using Siddhi in WSO2 DAS.

- [Creating a Standalone Execution Plan](#)
- [Creating a STORM Based Distributed Execution Plan](#)
- [Siddhi Query Language](#)

Creating a Standalone Execution Plan

WSO2 DAS uses execution plans to store event processing logic. An execution plan is bound to an instance of Siddhi Data Analytics Server runtime, which is responsible for the actual processing of events. DAS allows users to configure multiple execution plans and provides 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

Follow the instructions below to write an execution plan.

1. Start WSO2 DAS and log into its Management Console. Click **Main** and then click **Execution Plans** to open the **Available Execution Plans** page.
2. Click **Add Execution Plan** to open the **Create a New Execution Plan** page in which a template is displayed as shown in the example below.

Create a New Execution Plan

Enter Event Processor Details

Query Expressions

Import Stream*: Import Stream : As : Include Arbitrary :

Export Stream Value Of : StreamId : Include Arbitrary :

```

1 /* Enter a unique ExecutionPlan */
2 #@Plan:name('ExecutionPlan')
3
4 /* Enter a unique description for ExecutionPlan */
5 -- #@Plan:description('ExecutionPlan')
6
7 /* define streams/tables and write queries here ... */
8

```

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.

- a. Siddhi keywords, in alphabetical order
- b. 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.

Annotation	Description	Example
@plan:name (<name for execution plan>)	The name of the execution plan.	@Plan:name('ExecutionPlan')
@plan:description (<description about execution plan>)	This provides a description about the execution plan. Details such as the business requirement of the execution plan can be mentioned here.	@Plan:description('This is the description NewExecutionPlan')

@plan:async (bufferSize=<event queue size>)	<p>WSO2 DAS 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 1 024.</p>	@plan:async(bufferSize=2)
@async(bufferSize=<event queue size>)	<p>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.</p>	<pre>@async(bufferSize='2') define stream sensor_stream (meta_sensor humidity float, sensorValue double);</pre>
@import (<DAS stream definition>)	<p>This is used to import a DAS event stream definition and map it to a Siddhi stream definition. For more information about event stream definitions, see Understanding Event Streams and Event Tables.</p>	<pre>@Import('org.wso2.event.sensor.stream:1.0.0' define stream sensor_stream (meta_sensor humidity float, sensorValue double);</pre>

<code>@export (<DAS stream definition>)</code>	This is used to export a Siddhi event stream definition and map it to a DAS event stream definition.	<code>@Export('org.wso2.sensor.value.projected.stream')</code> <code>define stream sensor_value_projected</code> <code>(meta_sensorId int, humidity float, value double)</code>
------------------------------------------------------	------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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](#). This internal stream is meant to be used by the Siddhi runtime.

To import a stream:

1. Select an existing stream for the **Import Stream** parameter. This is the [event stream](#) from which events will be taken to be processed by the execution plan.
2. Enter a meaningful name for the stream in the **As** parameter. This is the name that is used when feeding the stream to the Siddhi engine. This can contain only alphanumeric characters and underscore (_).
3. Select the **Include Arbitrary** check box if you want to append arbitrary configurations to the stream definition when performing the import.

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.

4. Click **Import**.

This will import an available [event stream](#) as a new stream.

Step 3: Add query expressions

Query expressions are event processing logic written in Siddhi Query Language. When defining more than one query, each query should end with a semi colon.

```

1 /* Enter a unique ExecutionPlan */
2 #Plan:name('ExecutionPlan')
3
4 /* Enter a unique description for ExecutionPlan */
5 --- #Plan:description('ExecutionPlan')
6
7 /* define streams/tables and write queries here ... */
8
9
10 #Import('org.wso2.event.sensor.stream:1.0.0')
11 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);
12
13 #Export('org.wso2.event.sensor.filtered.stream:1.0.0')
14 define stream filteredStream (meta_timestamp long, meta_sensorName string, correlation_longitude double, correlation_latitude double, sensorValue double);
15
16 from sensorStream [sensorValue > 100]
17 select meta_timestamp, meta_sensorName, correlation_longitude, correlation_latitude, sensorValue
18 insert into filteredStream
19

```

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 ID of the [event stream](#) 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**.

Click **Add Execution Plan** 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 `<DAS_HOME>/repository/deployment/server/executionplans` directory.

Editing a deployed execution plan

Follow the procedure below to edit an execution plan.

- Start [WSO2 DAS](#) and log into 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.
- Click **Edit** in the row of the execution plan you want to edit. The **Edit Execution Plan Configuration** page will open.
- Edit the execution plan as required and click **Update**.

Alternatively, you can write your execution plan in a text file and save it with the `.siddhiql` extension (which stands for Siddhi Query Language), and place it in the `<PRODUCT_HOME>/repository/deployment`

nt/server/executionplans directory. Since hot deployment is supported you can also 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 WSO2 DAS and log into 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

WSO2 DAS uses storm based distributed execution plan to store the processing logic to be used in a [distributed mode deployment](#).

Writing an execution plan

The procedure for creating an execution plan is the same as that in [Creating a Standalone Execution Plan](#). In addition, the following annotations are used in the Siddhi queries.

Annotation	Description	Example
@dist (parallel='<number of Storm tasks>')	The number of storm tasks in which the query should be run parallel.	@dist(parallel='4')
@dist(execGroup='name of the group')	All the Siddhi queries in a particular execGroup will be executed in a single Siddhi bolt.	@dist(execGroup='Filtering')
@Plan:dist(receiverParallelism='number of receiver spouts')	The number of event receiver spouts to be spawned for the Storm topology.	@Plan:dist(receiverParallelism='1')
@Plan:dist(receiverParallelism='number of publisher spouts')	The number of event publisher spouts to be spawned for the Storm topology.	@Plan:dist(publisherParallelism='4')

The following execution plan is populated with the above mentioned annotations.

```
/* 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, its configuration in WSO2 Siddhi Query Language format is saved in the <DAS_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.

Siddhi Query Language

The guide provides specification of Siddhi Query Language 3.0 with examples

This guide provides instructions to use the Siddhi Query Language 3.0 with WSO2 DAS 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
- 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
 - RDBMS Event Table
 - Insert into
 - Delete
 - Update
 - Insert Overwrite
 - In
 - Join
- Event window
 - Event window definition
 - Sample event window definitions
 - Insert Into
 - Output
 - Join
- Event Trigger
 - Event Trigger Definition
- Siddhi Logger
- Siddhi Extensions
 - Extension Types
 - Function Extension
 - Aggregate Function Extension
 - Window Extension
 - Stream Function Extension
 - Stream Processor Extension
 - Available Extensions
 - Writing Custom Extensions

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.

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

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.

```
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.

Attribute Name	Attribute Type
deviceID	long
roomNo	int
temp	double

```
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.

```
from <input stream name>
select <attribute name>, <attribute name>, ...
insert into <output stream name>
```

e.g., If you want to derive only the room temperature from the `TempStream` event stream defined above, a query can be defined as follows.

```
from TempStream
select roomNo, temp
insert into RoomTempStream;
```

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.

Action	Description
Selecting required objects for projection	<p>This involves selecting only some of the attributes in an input stream to be inserted into an output stream.</p> <p>e.g., The following query selects only the roomNo and temp attributes from the TempStream stream.</p> <pre>from TempStream select roomNo, temp insert into RoomTempStream;</pre>
Selecting all attributes for projection	<p>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.</p> <p>e.g., Use one of the following queries to select all the attributes in the TempStream stream.</p> <pre>from TempStream select * insert into NewTempStream;</pre> <p>or</p> <pre>from TempStream insert into NewTempStream;</pre>
Renaming attributes	<p>This involves selecting attributes from the input streams and inserting them into the output stream with different names.</p> <p>e.g., The following query renames roomNo to roomNumber and temp to temperature .</p> <pre>from TempStream select roomNo as roomNumber, temp as temperature insert into RoomTempStream;</pre>
Introducing the default value	<p>This involves adding a default value and assigning it to an attribute using as.</p> <p>e.g.,</p> <pre>from TempStream select roomNo, temp, 'C' as scale insert into RoomTempStream;</pre>
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

Operator	Distribution	Example
()	Scope	(cost + tax) * 0.05
IS NULL	Null check	deviceID is null
NOT	Logical NOT	not (price > 10)
* / %	Multiplication, division, modulo	temp * 9/5 + 32
+ -	Addition, subtraction	temp * 9/5 + 32
< <=	Comparisons: less-than, greater-than-equal	
> >=	greater-than-equal, greater-than, less-than-equal	totalCost >= price * quantity
== !=	Comparisons: equal, not equal	totalCost >= price * quantity
IN	Contains in table	roomNo in ServerRoomsTable
AND	Logical AND	temp < 40 and (humidity < 40 or humidity >= 60)
OR	Logical OR	temp < 40 and (humidity < 40 or humidity >= 60)

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.

Unit	Syntax
Year	year years
Month	month months
Week	week weeks
Day	day days
Hour	hour hours
Minutes	minute minutes min
Seconds	second seconds sec
Milliseconds	millisecond milliseconds

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`

E.g. With convert and UUID function, converting room number to string and introducing message ID to each event.

```
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 condition should be defined in square brackets next to the input stream name.

```
from <input stream name>[<filter condition>]
select <attribute name>, <attribute name>, ...
insert into <output stream name>
```

E.g. Filtering all server rooms having temperature greater than 40 degrees.

```
from TempStream [(roomNo >= 100 and roomNo < 110) and temp > 40 ]
select roomNo, temp
insert into HighTempStream;
```

Window

Windows allows to capture a subset of events based on a criteria from input event stream for calculation, they can be defined next to input streams using '#window.' prefix and each input stream can only have maximum of one window as follows.

```
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`

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

E.g. Notify upon all event arrival and expiry the average temperature of all rooms based on all events arrived during last 10 minutes.

```
from TempStream#window.time(10 min)
select avg(temp) as avgTemp, roomNo, deviceID
insert all events into AvgTempStream;
```

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 words 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 emission can be specified.

With "<event interval>" the number of event need to be arrived for the periodic event emission 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 last temperature event per sensor every 10 events

```
from TempStream
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

```
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.

```
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)** will 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.

```
from <input stream name>[<filter condition>]#window.<window name>(<parameter>, ... )
{unidirectional} {as <reference>}
    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>
```

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 that's 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. Switch on temperature regulator if they are not already on, on all room which have current 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;
```

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.

```

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.

```

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).

```

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:> 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.

```
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 RDMS 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.

```

define table <table name> (<attribute name> <attribute type>, <attribute name>
<attribute type>, ... );

```

With the above definition events will store all events 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.

```

define table RoomTypeTable (roomNo int, type string);

```

Indexing Event Table

Event table can be index for fast event access using the "IndexedBy" annotation. With "IndexedBy" 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.

```

@IndexedBy('roomNo')
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 and the connection instructions can be assign 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.

```
@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 DAS server. To configure data sources in a WSO2 product, see [WSO2 Administration Guide - Managing Datasources](#).

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".

```
@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 RDMBS 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.

```
@From(eventtable='rdbms', datasource.name='AnalyticsDataSource',
table.name='RoomTable', cache='LRU', cache.size='3000')
define table RoomTypeTable (roomNo int, type string);
```

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

```
from TempStream
select *
insert into TempTable;
```

Delete

Query for deleting events on event table can be written using a delete query having following syntax

```
from <input stream name>
select <attribute name>, <attribute name>, ...
delete <table name>
on <condition>
```

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 that's 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, roomType string);

from UpdateStream
select roomType as type
delete 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

```

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.

```

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

```
<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:

Element	Description
<event window name>	A unique name for the window.
<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 .

<pre><output event type></pre>	<p>The possible values for this parameter are as follows:</p> <ul style="list-style-type: none"> • 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. <p>If the output event type is not specified for an event window, it emits both current and expired events.</p>
--------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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.

Keyword	Purpose
<code>current events</code>	To insert only current events into the event window.
<code>expired events</code>	To insert only expired events into the event window.
<code>all events</code>	To insert both current and expired events into the event window.

e.g., The following query inserts both current and expired events from an event stream named `sensorStream` to an event window named `SensorWindow`.

```
from SensorStream
insert into 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`.

```
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.

```
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.

Parameter	Description
priority	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.
logMessage	This parameter allows you to specify a message to be printed in the log.
isEventLogged	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.

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;
```

Siddhi Extensions

Siddhi support an extension architecture to support custom code and functions to be incorporated with Siddhi in a seamless manner. Extension will follows the 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 to 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 geo coordinates
- **r**: Supporting R executions
- **regex**: Supporting regular expression operations

Extensions released under GNU/GPL License v3 :

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 interferes. E.g implementing a Unique Window Processor by extending org.wso2.siddhi.core.query.processor.stream.window.WindowProcessor.

```
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;

```
# function name to class mapping of 'foo' extension
unique=org.wso2.test.UniqueWindowProcessor
```

Refer following for implementing different types of Siddhi extensions with examples

- Function Extension
 - Aggregate Function Extension
 - Window Extension
 - Stream Function Extension
 - Stream Processor Extension
-

- Inbuilt Functions
- Inbuilt Windows
- Inbuilt Aggregate Functions
- Writing Extensions to Siddhi
- Siddhi Extensions
- Using Siddhi as a Library

Inbuilt Functions

Following are the supported inbuilt functions of Siddhi

- coalesce
- convert
- instanceOfBoolean
- instanceOfDouble
- instanceOfFloat
- instanceOfInteger
- instanceOfLong
- instanceOfString
- UUID

coalesce

```
< 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 non null input parameter.
- **Parameters:** This function accepts one or more parameters, and they all have to be the same type of, any one of the available types.
- **Return Type:** Return type will be the first input parameter's type.
- **Examples:** coalesce('123', null, '789') returns '123'. coalesce(null, 76, 567) returns 76. coalesce(null, null, null) returns null.

convert

```
< int|long|float|double|string|bool > convert (<int|long|float|double|string|bool> t oBeConverted, <string> convertedTo)
```

- **Extension Type:** Function

- **Description:** Converts the first input parameter according to the convertedTo parameter.
- **Parameter: toBeConverted :** To be converted parameter with type other than object.
- **Parameter: 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:** Return type will be 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

< bool > **instanceOfBoolean** (<int|long|float|double|string|bool|object> **arg**)

- **Extension Type:** Function
- **Description:** Checks if the parameter is an instance of Boolean or not.
- **Parameter: arg :** The parameter to be checked.
- **Return Type:** Returns bool, true if the parameter is an instance of Boolean and false otherwise.
- **Examples:** instanceOfBoolean(123) returns false. instanceOfBoolean(true) returns true. instanceOfBoolean(false) returns true.

instanceOfDouble

< bool > **instanceOfDouble** (<int|long|float|double|string|bool|object> **arg**)

- **Extension Type:** Function
- **Description:** Checks if the parameter is an instance of Double or not.
- **Parameter: arg :** The parameter to be checked.
- **Return Type:** Returns bool, true if the parameter is an instance of Double and false otherwise.
- **Examples:** instanceOfDouble(123) returns false. instanceOfDouble(56.45) returns true. instanceOfDouble(false) returns false.

instanceOfFloat

< bool > **instanceOfFloat** (<int|long|float|double|string|bool|object> **arg**)

- **Extension Type:** Function
- **Description:** Checks if the parameter is an instance of Float or not.
- **Parameter: arg :** The parameter to be checked.
- **Return Type:** Returns bool, true if the parameter is an instance of Float and false otherwise.
- **Examples:** instanceOfFloat(123) returns false. instanceOfFloat(56.45) returns false. instanceOfFloat(56.45f) returns true.

instanceOfInteger

< bool > **instanceOfInteger** (<int|long|float|double|string|bool|object> **arg**)

- **Extension Type:** Function
- **Description:** Checks if the parameter is an instance of Integer or not.
- **Parameter: arg :** The parameter to be checked.
- **Return Type:** Returns bool, true if the parameter is an instance of Integer and false otherwise.
- **Examples:** instanceOfInteger(123) returns true. instanceOfInteger(56.45) returns false. instanceOfInteger(56.45f) returns false.

instanceOfLong

< bool > **instanceOfLong** (<int|long|float|double|string|bool|object> **arg**)

- **Extension Type:** Function
- **Description:** Checks if the parameter is an instance of Long or not.
- **Parameter: arg :** The parameter to be checked.
- **Return Type:** Returns bool, true if the parameter is an instance of Long and false otherwise.
- **Examples:** instanceOfLong(123) returns false. instanceOfLong(56671) returns true. instanceOfLong(56.67) returns false.

instanceOfString

< bool > **instanceOfString** (<int|long|float|double|string|bool|object> **arg**)

- **Extension Type:** Function

- **Description:** Checks if the parameter is an instance of String or not.
- **Parameter:** **arg** : The parameter to be checked.
- **Return Type:** Returns bool, true if the parameter is an instance of String and false otherwise.
- **Examples:** `instanceOfString('test')` returns true. `instanceOfString('5667')` returns true. `instanceOfString(56.67)` returns false.

UUID

< string > **UUID** ()

- **Extension Type:** Function
- **Description:** Generate a UUID.
- **Return Type:** Returns a UUID string.
- **Examples:** `UUID()` returns a34eec40-32c2-44fe-8075-7f4fde2e2dd8.

E.g. Converting room number to string and introducing message ID to each event

```
from TempStream
select convert(roomNo, 'string') as roomNo, temp, UUID() as messageID
insert into RoomTempStream;
```

Inbuilt Windows

Following are the supported inbuilt windows of Siddhi

- `time`
- `timeBatch`
- `length`
- `lengthBatch`
- `externalTime`
- `cron`
- `firstUnique`
- `unique`
- `sort`
- `frequent`
- `lossyFrequent`
- `externalTimeBatch`
- `timeLength`

time

Syntax	<code><event> time(<int long time> windowTime)</code>
Extension Type	Window
Description	A sliding time window that holds events that arrived during the last <code>windowTime</code> period at a given time, and gets updated for each event arrival and expiry.
Parameter	<ul style="list-style-type: none"> • <code>windowTime</code> : The sliding time period for which the window should hold events.
Return Type	Returns current and expired events.
Examples	<ul style="list-style-type: none"> • <code>time(20)</code> for processing events that arrived within the last 20 milliseconds. • <code>time(2 min)</code> for processing events that arrived within the last 2 minutes.

timeBatch

Syntax	<code><event> timeBatch(<int long time> windowTime)</code>
Extension Type	Window
Description	A batch (tumbling) time window that holds events that arrive during <code>windowTime</code> periods, and gets updated for each <code>windowTime</code> .
Parameter	<code>windowTime</code> : The batch time period for which the window should hold events.
Return Type	Returns current and expired events.
Examples	<ul style="list-style-type: none"> • <code>timeBatch(20)</code> for processing events that arrive every 20 milliseconds. • <code>timeBatch(2 min)</code> for processing events that arrive every 2 minutes.

length

Syntax	<code><event> length(<int> windowLength)</code>
Extension Type	Window
Description	A sliding length window that holds the last <code>windowLength</code> events at a given time, and gets updated for each arrival and expiry.
Parameter	<ul style="list-style-type: none"> • <code>windowLength</code> : The number of events that should be included in a sliding length window.
Return Type	Returns current and expired events.
Examples	<ul style="list-style-type: none"> • <code>length(10)</code> for processing the last 10 events. • <code>length(200)</code> for processing the last 200 events.

lengthBatch

Syntax	<code><event> lengthBatch(<int> windowLength)</code>
Extension Type	Window
Description	A batch (tumbling) length window that holds a number of events specified as the <code>windowLength</code> . The window is updated each time a batch of events that equals the number specified as the <code>windowLength</code> arrives.
Parameter	<code>windowLength</code> : The number of events the window should tumble.
Return Type	Returns current and expired events.
Examples	<ul style="list-style-type: none"> • <code>lengthBatch(10)</code> for processing 10 events as a batch. • <code>lengthBatch(200)</code> for processing 200 events as a batch.

externalTime

Syntax	<code><event> externalTime(<long> timestamp, <int long time> windowTime)</code>
---------------	---------------------------------------------------------------------------------------------------

Extension Type	Window
Description	A sliding time window based on external time. It holds events that arrived during the last <code>windowTime</code> period from the external timestamp, and gets updated on every monotonically increasing timestamp.
Parameter	<ul style="list-style-type: none"> • <code>windowTime</code> : The sliding time period for which the window should hold events.
Return Type	Returns current and expired events.
Examples	<ul style="list-style-type: none"> • <code>externalTime(eventTime, 20)</code> for processing events arrived within the last 20 milliseconds from the <code>eventTime</code> • <code>externalTime(eventTimestamp, 2 min)</code> for processing events arrived within the last 2 minutes from the <code>eventTimestamp</code>

cron

Syntax	<code><event> cron(<string> cronExpression)</code>
Extension Type	Window
Description	This window returns events processed periodically as the output in time-repeating patterns, triggered based on time passing.
Parameter	<code>cronExpression</code> : cron expression that represents a time schedule.
Return Type	Returns current and expired events.
Examples	<code>cron('*/5 * * * * ?')</code> returns processed events as the output every 5 seconds.

firstUnique

Syntax	<code><event> firstUnique(<string> attribute)</code>
Extension Type	Window
Description	First unique window processor keeps only the first events that are unique according to the given unique attribute.
Parameter	<code>attribute</code> : The attribute that should be checked for uniqueness.
Return Type	Returns current and expired events.
Examples	<code>firstUnique(ip)</code> returns the first event arriving for each unique ip.

unique

Syntax	<code><event> unique (<string> attribute)</code>
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	<code>unique(ip)</code> returns the latest event that arrives for each unique ip.

sort

Syntax	<pre><event> sort(<int> windowLength)</pre> <pre><event> sort(<int> windowLength, <string> attribute, <string> order)</pre> <pre><event> sort(<int> windowLength, <string> attribute, <string> order, ... , <string> attributeN, <string> orderN)</pre>
Extension Type	Window
Description	This window holds a batch of events that equal the number specified as the <code>windowLength</code> and sorts them in the given order.
Parameter	<ul style="list-style-type: none"> • attribute : The attribute that should be checked for the order.
Return Type	Returns current and expired events.
Examples	<code>sort(5, price, 'asc')</code> keeps the events sorted by price in the ascending order. Therefore, at any given time, the window contains the 5 lowest prices.

frequent

Syntax	<pre><event> frequent(<int> eventCount)</pre> <pre><event> frequent(<int> eventCount, <string> attribute, ... , <string> attributeN)</pre>
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	<ul style="list-style-type: none"> • eventCount : The number of most frequent events to be emitted to the stream.
Return Type	Returns current and expired events.
Examples	<ul style="list-style-type: none"> • <code>frequent(2)</code> returns the 2 most frequent events. • <code>frequent(2, cardNo)</code> returns the 2 latest events with the most frequently appeared card numbers.

lossyFrequent

Syntax	<pre><event> lossyFrequent(<double> supportThreshold, <double> errorBound) <event> lossyFrequent(<double> supportThreshold, <double> errorBound, <string> attribute, . . . , <string> attributeN)</pre>
Extension Type	Window
Description	This window identifies and returns all the events of which the current frequency exceeds the value specified for the <code>supportThreshold</code> parameter.
Parameters	<ul style="list-style-type: none"> • <code>errorBound</code> : The error bound value. • <code>attribute</code> : 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	<ul style="list-style-type: none"> • <code>lossyFrequent(0.1, 0.01)</code> returns all the events of which the current frequency exceeds 0.1, with an error bound of 0.01. • <code>lossyFrequent(0.3, 0.05, cardNo)</code> returns all the events of which the <code>cardNo</code> attributes frequency exceeds 0.3, with an error bound of 0.05.

externalTimeBatch

Syntax	<pre><event> externalTimeBatch(<long> timestamp, <int long time> windowTime)</pre>
Extension Type	Window
Description	A batch (tumbling) time window based on external time, that holds events arrived during <code>windowTime</code> periods, and gets updated for every <code>windowTime</code> .
Parameters	<ul style="list-style-type: none"> • <code>timestamp</code> : The time which the window determines as current time and will act upon. The value of this parameter should be monotonically increasing. • <code>windowTime</code> : The batch time period for which the window should hold events.
Return Type	Returns current and expired events.
Examples	<ul style="list-style-type: none"> • <code>externalTimeBatch(eventTime, 20)</code> for processing events that arrive every 20 milliseconds from the <code>eventTime</code>. • <code>externalTimeBatch(eventTimestamp, 2 min)</code> for processing events that arrive every 2 minutes from the <code>eventTimestamp</code>.

timeLength

Syntax	<pre><event> timeLength (< int long time > windowTime, < int > windowLength)</pre>
Extension Type	Window
Description	A sliding time window that, at a given time holds the last <code>windowLength</code> events that arrived during last <code>windowTime</code> period, and gets updated for every event arrival and expiry.

Parameters	<ul style="list-style-type: none"> • <code>windowTime</code>: The sliding time period for which the window should hold events. • <code>windowLength</code>: The number of events that should be included in a sliding length window.
Return Type	Returns current and expired events.
Examples	<ul style="list-style-type: none"> • <code>timeLength(20 sec, 10)</code> for processing the last 10 events that arrived within the last 20 seconds. • <code>timeLength(2 min, 5)</code> for processing the last 5 events that arrived within the last 2 minutes.

Inbuilt Aggregate Functions

Following are the supported inbuilt aggregate functions of Siddhi

- `sum`

`sum`

`<long|double> sum (<int|long|double|float> arg)`

- **Extension Type:** Aggregate Function
- **Description:** Sums all the events.
- **Parameter:** `arg`: The value that need 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:** `sum(20)` returns sum of 20s as a long value for each event arrival and expiry. `sum(temp)` returns the sum of all temp attributes based on each event arrival and expiry.

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 Window Extension](#)
- [Writing a Custom Aggregate Function](#)
- [Writing a Custom Stream Function Extension](#)
- [Writing a Custom Function](#)
- [Writing a Custom Stream Processor Extension](#)

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 DAS, add the jar to the `<DAS_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.

```
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.

```
/*
 * 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.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 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 DAS, add it to the `<Das_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;

```

/*
 * 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.AttributeAggregator;
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;

```
#  
# 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.  
#  
std=org.wso2.siddhi.core.query.selector.attribute.aggregator.StrandedDeviationAggregat  
eFunction
```

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.

1. Create a class extending `org.wso2.siddhi.core.query.processor.stream.function.StreamFunctionProcessor`.
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 DAS, add the jar to the `<DAS_HOME>/repository/components/dropins` 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.

```
/*  
 * 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.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.StreamFunctionProcessor;
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 Geocoder();
    private boolean debugModeOn;

    /**
     * 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) {
}

```

```
//Implement restore state logic.
}
}
```

Sample geo.siddhiext extension mapping file for the custom stream 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.  
  
geocode=org.wso2.siddhi.extension.geo.GeocodeStreamFunctionProcessor
```

Writing a Custom Function

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 DAS, add it to the `<Das_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.

```
from InValueStream
select math:sin(inValue) as sinValue
insert into OutMediationStream;
```

Note

From CEP 3.0.0 onwards, `FunctionExecutor` is supposed to be used for writing both custom expressions and conditions.

The following is a sample implementation of a custom function extension.

```

/*
 * 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.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.definition.Attribute;
import org.wso2.siddhi.query.api.exception.ExecutionPlanValidationException;

/*
 * sin(a);
 * Returns the sine of a (a is in radians).
 * Accept Type(s) :DOUBLE/INT/FLOAT/LONG
 * Return Type(s): DOUBLE
 */
public class SinFunctionExtension extends FunctionExecutor {

    /**
     * The initialization method for SinFunctionExtension, this method will be called
     * before the other methods
     *
     * @param attributeExpressionExecutors the executors of each function parameter
     * @param executionPlanContext          the context of the execution plan
     */
    @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);
        } 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
     *
     * @return stateful objects of the processing element as an array
     */
    @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 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 DAS, add the jar to the `<DAS_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.

```

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.

```

/*
 * 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.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.StreamEventCloner;
import org.wso2.siddhi.core.event.stream.populator.ComplexEventPopulator;
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 confidenceInterval,
int/long/float/double y, int/long/float/double x1, int/long/float/double x2 ...)
 */
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) {
    paramInt = attributeExpressionLength;

    // Capture constant inputs
    if (attributeExpressionExecutors[0] instanceof ConstantExpressionExecutor){
        paramInt = paramInt - 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 (paramInt > SIMPLE_LINREG_INPUT_PARAM_COUNT) {
        regressionCalculator = new MultipleLinearRegressionCalculator(paramInt,
calcInterval, batchSize, ci);
    } else {
        regressionCalculator = new SimpleLinearRegressionCalculator(paramInt,
calcInterval, batchSize, ci);
    }

    // Add attributes for standard error and all beta values
    String betaVal;
    ArrayList<Attribute> attributes = new ArrayList<Attribute>(paramInt);
    attributes.add(new Attribute("stderr", Attribute.Type.DOUBLE));

    for (int itr = 0; itr < paramInt; 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 complexEventPopulator helps to populate the events with the resultant
attributes
 */
@Override
protected void process(ComplexEventChunk<StreamEvent> streamEventChunk, Processor
nextProcessor, StreamEventCloner streamEventCloner, ComplexEventPopulator
complexEventPopulator) {
    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 {
            complexEventPopulator.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;

```
#  
# 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.LinearRegressionStreamProcessor
```

Siddhi Extensions

Siddhi language has a wide verity of extension 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.

```
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.

- [math](#)
- [str](#)
- [geo](#)

- r
- regex
- time
- nlp
- pmml
- ml
- timeseries
- kf (Kalman Filter)
- map
- reorder

math

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.

abs

Syntax	<code><double> abs(<float double> p1)</code>
Extension Type	Function
Description	Returns the absolute value of <code>p1</code> . This function wraps the <code>java.lang.Math.abs()</code> function.
Examples	Both the following queries return 3 since the absolute value of both 3 and -3 is 3. <ul style="list-style-type: none"> • <code>abs(3)</code> • <code>abs(-3)</code>

acos

Syntax	<code><double> acos(<float double> p1)</code>
Extension Type	Function
Description	If $-1 \leq p1 \leq 1$, this function returns the arc-cosine (inverse cosine) of <code>p1</code> . If not, it returns <code>NULL</code> . The return value is in radian scale. This function wraps the <code>java.lang.Math.acos()</code> function.
Example	<code>acos(0.5)</code> returns 1.0471975511965979.

asin

Syntax	<code><double> asin (<float double> p1)</code>
Extension Type	Function
Description	If $-1 \leq p1 \leq 1$, this function returns the arc-sin (inverse sine) of <code>p1</code> . If not, it returns <code>NULL</code> . The return value is in radian scale. This function wraps the <code>java.lang.Math.asin()</code> function.
Example	<code>asin(0.5)</code> returns 0.5235987755982989.

atan

Syntax	<code><double> atan(<int long float double> p1)</code>
---------------	--------------------------------------------------------------------

Extension Type	Function
Description	Returns the arc-tangent (inverse tangent) of p1 . The return value is in radian scale. This function wraps the <code>java.lang.Math.atan()</code> function.
Examples	<code>atan(6d)</code> returns <code>1.4056476493802699.</code>

Syntax	<code><double> atan (<int long float double> p1, <int long float double> p2)</code>
Extension Type	Function
Description	Returns the arc-tangent (inverse tangent) of p1 and p2 coordinates. The return value is in radian scale. This function wraps the <code>java.lang.Math.atan2()</code> function.
Examples	<code>atan(12d, 5d)</code> returns <code>1.1760052070951352.</code>

bin

Syntax	<code><string> bin(<int long> p1)</code>
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 <code>java.lang.Integer.toBinaryString</code> and <code>java.lang.Long.toBinaryString</code> methods.
Examples	<code>bin(9)</code> returns <code>"1001".</code>

ceil

Syntax	<code><double> ceil(<float double> p1)</code>
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 <code>java.lang.Math.ceil()</code> method.
Example	<code>ceil(423.187d)</code> returns <code>424.0.</code>

conv

Syntax	<code><string> conv(<string> a, <int> fromBase, <int> toBase)</code>
Extension Type	Function
Description	Converts a from the fromBase base to the toBase base.
Example	<code>conv("7f", 16, 10)</code> returns <code>"127".</code>

copySign

Syntax	<code><double> copySign(<int long float double> magnitude, <int long float double> sign)</code>
---------------	-------------------------------------------------------------------------------------------------------------------

Extension Type	Function
Description	Returns the magnitude of <code>magnitude</code> with the sign of <code>sign</code> . This function wraps the <code>java.lang.Math.copySign()</code> function.
Example	<code>copySign(5.6d, -3.0d)</code> returns <code>-5.6</code> .

cos

Syntax	<code><double> cos(<int long float double> p1)</code>
Extension Type	Function
Description	Returns the cosine of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.cos()</code> function.
Example	<code>cos(6d)</code> returns <code>0.9601702866503661</code> .

cosh

Syntax	<code><double> cosh(<int long float double> p1)</code>
Extension Type	Function
Description	Returns the hyperbolic cosine of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.cosh()</code> function.
Example	<code>cosh(6d)</code> returns <code>201.7156361224559</code> .

cbrt

Syntax	<code><double> cbrt(<int long float double> p1)</code>
Extension Type	Function
Description	Returns the cube-root of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.cbrt()</code> function.
Example	<code>cbrt(17d)</code> returns <code>2.5712815906582356</code> .

e

Syntax	<code><double> e()</code>
Extension Type	Function
Description	Returns the <code>java.lang.Math.E</code> constant, which is the closest double value to <code>e</code> (which is the base of the natural logarithms).
Example	<code>e()</code> returns <code>2.7182818284590452354</code> .

exp

Syntax	<code><double> exp(<int long float double> p1)</code>
Extension Type	Function
Description	Returns Euler's number <code>e</code> raised to the power of <code>p1</code> . This function wraps the <code>java.lang.Math.exp()</code> function.
Example	<code>exp(10.23)</code> returns <code>27722.51006805505</code> .

floor

Syntax	<code><double> floor(<int long float double> p1)</code>
Extension Type	Function
Description	This function wraps the <code>java.lang.Math.floor()</code> function that returns the largest (closest to positive infinity) value that is less than or equal to <code>p1</code> , and is equal to a mathematical integer.
Example	<code>floor(10.23)</code> returns <code>10.0</code> .

getExponent

Syntax	<code><double> getExponent(<int long float double> p1)</code>
Extension Type	Function
Description	Returns the unbiased exponent used in the representation of <code>p1</code> . This function wraps the <code>java.lang.Math.getExponent()</code> function.
Example	<code>getExponent(60984.1)</code> returns <code>15</code> .

hex

Syntax	<code><string> hex(<int long float double> p1)</code>
Extension Type	Function
Description	This function wraps the <code>java.lang.Double.toHexString()</code> function that returns a hexadecimal string representation of <code>p1</code> .
Example	<code>hex(200)</code> returns <code>"c8"</code> .

isInfinite

Syntax	<code><boolean> isInfinite(<float double> p1)</code>
Extension Type	Function
Description	This function wraps the <code>java.lang.Float.isInfinite()</code> and <code>java.lang.Double.isInfinite()</code> functions that return <code>true</code> if <code>p1</code> is infinitely large in magnitude, or return <code>false</code> otherwise.
Example	<code>isInfinite(java.lang.Double.POSITIVE_INFINITY)</code> returns <code>true</code> .

isNaN

Syntax	< boolean> isNaN (<float double> p1)
Extension Type	Function
Description	This function wraps the <code>java.lang.Float.isNaN()</code> and <code>java.lang.Double.isNaN()</code> functions that return <code>true</code> if <code>p1</code> is a NaN (Not-a-Number) value, or return <code>false</code> otherwise.
Example	<code>isNaN(java.lang.Math.log(-12d))</code> returns <code>true</code> .

ln

Syntax	<double> ln (< int long float double > p1)
Extension Type	Function
Description	Returns the natural logarithm (base e) of <code>p1</code> .
Example	<code>ln(11.453)</code> returns <code>2.438251704415579</code> .

log2

Syntax	<double> log2 (< int long float double > p1)
Extension Type	Function
Description	Returns the base 2 logarithm of <code>p1</code> .
Example	<code>log2(91d)</code> returns <code>6.507794640198696</code> .

log10

Syntax	<double> log10 (< int long float double > p1)
Extension Type	Function
Description	Returns the base 10 logarithm of <code>p1</code> .
Example	<code>log10(19.234)</code> returns <code>1.2840696117100832</code> .

log

Syntax	<double> log (< int long float double > number , < int long float double > base)
Extension Type	Function
Description	Returns the logarithm (base=base) of <code>number</code> .
Example	<code>log(34, 2f)</code> returns <code>5.08746284125034</code> .

max

Syntax	<double> max (< int long float double > p1 , < int long float double > p2)
---------------	--------------------------------------------------------------------------------------------------

Extension Type	Function
Description	Returns the greater value out of <code>p1</code> and <code>p2</code> .
Example	<code>max(123.67d, 91)</code> returns 123.67.

min

Syntax	<code><double> min (< int long float double > p1, <int long float double> p2)</code>
Extension Type	Function
Description	Returns the smaller value out of <code>p1</code> and <code>p2</code> .
Example	<code>min(123.67d, 91)</code> returns 91.

oct

Syntax	<code><string> oct (<int long> p1)</code>
Extension Type	Function
Description	Converts <code>p1</code> to octal.
Example	<code>oct(991)</code> returns "143".

parseDouble

Syntax	<code><double> parseDouble (<string> str)</code>
Extension Type	Function
Description	Returns <code>str</code> as a double.
Example	<code>parseDouble("123")</code> returns 123.0.

parseFloat

Syntax	<code><float> parseFloat (<string> str)</code>
Extension Type	Function
Description	Returns <code>str</code> as a float.
Example	<code>parseFloat("123")</code> returns 123.0.

parseInt

Syntax	<code><int> parseInt (<string> str)</code>
Extension Type	Function
Description	Returns <code>str</code> as an int.
Example	<code>parseInt("123")</code> returns 123.

parseLong

Syntax	<code><long> parseLong (<string> str)</code>
Extension Type	Function
Description	Returns <code>str</code> as a long.
Example	<code>parseLong("123")</code> returns 123.

pi

Syntax	<code><double> pi ()</code>
Extension Type	Function
Description	Returns the <code>java.lang.Math.PI</code> constant, which is the closest value to pi (i.e. the ratio of the circumference of a circle to its diameter).
Example	<code>pi()</code> always returns 3.141592653589793.

power

Syntax	<code><double> power (< int long float double> value, <int long float double> toPower)</code>
Extension Type	Function
Description	Returns <code>value</code> raised to the power of <code>toPower</code> .
Example	<code>power(5.6d, 3.0d)</code> returns 175.61599999999996.

rand

Syntax	<code><double> rand ()</code>
Extension Type	Function
Description	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.
Example	Two sequential calls to <code>rand()</code> may return 0.8263929447650588 and 0.24425883860361197 respectively.

Syntax	<code><double> rand (< int long > seed)</code>
Extension Type	Function
Description	A sequence of calls to <code>rand(seed)</code> generates a stream of pseudo-random numbers. This function uses the <code>java.util.Random</code> class internally.
Example	Two sequential calls to <code>rand(12)</code> may return 0.7298928061101974 and 0.2750691655200749, respectively.

round

Syntax	<code><int> round (<float> value)</code>
Extension Type	Funcion
Description	Returns the closest integer value to the argument.
Example	round(3.35) returns 3.

Syntax	<code><long> round (<double> value)</code>
Extension Type	Function
Description	Returns the closest long value to the argument.
Example	round(3252.353) returns 3252.

signum

Syntax	<code><int> signum (< int long float double > p1)</code>
Extension Type	Function
Description	<ul style="list-style-type: none"> 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. <p>This function wraps the <code>java.lang.Math.signum()</code> function.</p>
Example	signum(-6.32d) returns -1.

sin

Syntax	<code><double> sin (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the sine of p1 (p1 is in radians). This function wraps the <code>java.lang.Math.sin()</code> function.
Example	sin(6d) returns -0.27941549819892586.

sinh

Syntax	<code><double> sinh (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the hyperbolic sine of p1 (p1 is in radians). This function wraps the <code>java.lang.Math.sinh()</code> function.
Example	sinh(6d) returns 201.71315737027922.

sqrt

Syntax	<code><double> sqrt (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the square-root of <code>p1</code> . This function wraps the <code>java.lang.Math.sqrt()</code> function.
Example	<code>sqrt(4d)</code> returns 2.

tan

Syntax	<code><double> tan (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the tan of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.tan()</code> function.
Example	<code>tan(6d)</code> returns -0.29100619138474915.

tanh

Syntax	<code><double> tanh (<int long float double> p1)</code>
Extension Type	Function
Description	Returns the hyperbolic tangent of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.tanh()</code> function.
Example	<code>tanh(6d)</code> returns 0.9999877116507956.

toDegrees

Syntax	<code><double> toDegrees (< int long float double > p1)</code>
Extension Type	Function
Description	Converts <code>p1</code> from radians to degrees. This function wraps the <code>java.lang.Math.toDegrees()</code> function.
Example	<code>toDegrees(6d)</code> returns 343.77467707849394.

toRadians

Syntax	<code><double> toRadians (< int long float double > p1)</code>
Extension Type	Function
Description	Converts <code>p1</code> from degrees to radians. This function wraps the <code>java.lang.Math.toRadians()</code> function.
Example	<code>toRadians(6d)</code> returns 0.10471975511965977.

str

This extension provides basic string handling capabilities such as con-cat, length, convert to lowercase, replace all, etc. Following are the functions of the String extension.

charAt

Syntax	<string> charAt (<string> str , <int> index)
Extension Type	Function
Description	Returns the char value as a string value at the specified index.
Example	<code>charAt("WSO2" , 1)</code> returns 'S'.

coalesce

Syntax	< int long float double string boolean > coalesce (< 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 the value of the first of its input parameters that is not null.
Parameters	This functions 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	<ul style="list-style-type: none"> • <code>coalesce("123", null, "789")</code> returns "123". • <code>coalesce(null, "BBB", "CCC")</code> returns "BBB". • <code>coalesce(null, null, null)</code> returns null.

concat

Syntax	<string> 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	<ul style="list-style-type: none"> • <code>concat("D533", "8JU^", "XYZ")</code> returns "D5338JU^XYZ". • <code>concat("AAA", null, "CCC")</code> returns "AAACCC".

hex

Syntax	<string> hex (< string> str)
Extension Type	Function
Description	Returns a hexadecimal string representation of str .
Example	<code>hex("MySQL")</code> returns "4d7953514c".

length

Syntax	<code><int> length (< string> str)</code>
Extension Type	Function
Description	Returns the length of the string: <code>str</code> .
Examples	<code>length("Hello World")</code> returns 11.

lower

Syntax	<code><string> lower (< string> str)</code>
Extension Type	Function
Description	Converts the capital letters in the <code>str</code> input string to the equivalent simple letters.
Example	<code>lower("WSO2 cep")</code> returns "wso2 cep".

regexp

Syntax	<code><boolean> regexp (< string> str, <string> regex)</code>
Extension Type	Function
Description	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.
Example	<code>regexp("WSO2 abcdh" , "WSO(.*)h")</code> returns <code>true</code> .

repeat

Syntax	<code><string> repeat (< string> str, <int> times)</code>
Extension Type	Function
Description	Repeats the specified string (i.e. <code>str</code>) for the specified number of times (i.e. <code>times</code>).
Example	<code>repeat("StRing 1" , 3)</code> returns "StRing 1StRing 1StRing 1".

replaceAll

Syntax	<code><string> replaceAll (< string> str, <string> regex , <string> replacement)</code>
Extension Type	Function
Description	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>).
Example	<code>replaceAll("hello hi hello" , 'hello' , 'test')</code> returns "test hi test".

replaceFirst

Syntax	<code><string> replaceFirst (< string> str , <string> regex , <string> replacement)</code>
---------------	---------------------------------------------------------------------------------------------------------------------

Extension Type	Function
Description	Replaces the first substring that matches the given regular expression (i.e. <code>regex</code>) with the string specified as the replacement (i.e. <code>replacement</code>)
Example	<code>replaceFirst("hello WSO2 A hello", 'WSO2(.*)A', 'XXXX') returns "hello XXXX hello".</code>

reverse

Syntax	<code><string> reverse (< string> str)</code>
Extension Type	Function
Description	Returns the reverse ordered string of <code>str</code> .
Example	<code>reverse("Hello World") returns "dlrow olleH".</code>

strcmp

Syntax	<code><int> strcmp (< string> str, <string> compareTo)</code>
Extension Type	Function
Description	Compares <code>str</code> with <code>compareTo</code> strings lexicographically.
Examples	<ul style="list-style-type: none"> <code>strcmp("Hello", 'Hello')</code> returns 0. <code>strcmp("AbCDefghiJ KLMN", 'Hello')</code> returns -7.

substr

Syntax	<code><string> substr (< string> sourceText, <int> beginIndex)</code>
Extension Type	Function
Description	Returns a new string that is a substring of <code>sourceText</code> .
Example	<code>substr("AbCDefghiJ KLMN", 4) returns "efghiJ KLMN".</code>

Syntax	<code><string> substr (< string> sourceText, <int> beginIndex, <int> length)</code>
Extension Type	Function
Description	Returns a new string that is a substring of <code>sourceText</code> .
Example	<code>substr("AbCDefghiJ KLMN", 2, 4) returns "CDef".</code>

Syntax	<code><string> substr (< string> sourceText, <string> regex)</code>
Extension Type	Function
Description	Returns a new string that is a substring of <code>sourceText</code> .

Examples	<code>substr("WSO2D efghiJ KLMN", '^WSO2(.*)') returns "WSO2D efghiJ KLMN".</code>
-----------------	------------------------------------------------------------------------------------

Syntax	<code><string> substr (< string> sourceText, <string> regex, <int> groupNumber)</code>
Extension Type	Function
Description	Returns a new string that is a substring of <code>sourceText</code> .
Example	<code>substr("WSO2 cep WSO2 XX E hi hA WSO2 heAllo", 'WSO2(.*)A(.*)', 2) returns " ello".</code>

trim

Syntax	<code><string> trim (< string> str)</code>
Extension Type	Function
Description	Returns a copy of <code>str</code> , with the leading and/or trailing white-spaces omitted.
Example	<code>trim(" AbCDefghiJ KLMN ") returns "AbCDefghiJ KLMN".</code>

unhex

Syntax	<code><string> unhex (< string> str)</code>
Extension Type	Function
Description	This is the equivalent of the <code>unhex</code> function in mysql 5.0. <code>unhex(str)</code> interprets each pair of characters in <code>str</code> as a hexadecimal number. Also see hex () string extension.
Example	<code>unhex("4d7953514c") returns "MySQL".</code>

upper

Syntax	<code><string> upper (<string> str)</code>
Extension Type	Function
Description	Converts the simple letters in the given input string (i.e. <code>str</code>) to the equivalent capital letters.
Example	<code>upper("Hello World") returns "HELLO WORLD".</code>

contains

Syntax	<code><bool> contains (< string> inputSequence, <string> searchingSequence)</code>
Extension Type	Function
Description	This method returns true if <code>inputSequence</code> contains the specified sequence of char values in the <code>searchingSequence</code> .
Example	<code>contains("21 products are produced by WSO2 currently", "WSO2") returns true.</code>

geo

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

Syntax	<code><bool> intersects (<string> geoJSONGeometry , <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns <code>true</code> if the geoJSONGeometry incoming event intersects the given string (i.e. geoJSONGeometryFence). Returns <code>false</code> otherwise.
Example	<code>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]]] })</code> , returns <code>true</code> because geoJSONGeometry intersects geoJSONGeometryFence.

Syntax	<code><bool> intersects (<double> longitude , <double> latitude , <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns <code>true</code> if the location specified in terms of longitude and latitude intersects the given geoJSONGeometryFence . Returns <code>false</code> otherwise.
Example	<code>intersects(0.5, 0.5 , { 'type':'Polygon', 'coordinates':[[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]] })</code> , returns <code>true</code> because the location specified in terms of longitude and latitude intersects geoJSONGeometryFence.

within

Syntax	<code><bool> within (<double> longitude , <double> latitude, <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns <code>true</code> if the location specified in terms of longitude and latitude is within the geoJSONGeometryFence.
Examples	<ul style="list-style-type: none"> <code>within(0.5, 0.5 , { 'type':'Polygon', 'coordinates':[[[0,0],[0,2],[1,2],[1,0],[0,0]]] })</code> returns <code>true</code>. <code>within(2, 2 , { 'type':'Polygon', 'coordinates':[[[0,0],[0,2],[1,2],[1,0],[0,0]]] })</code> returns <code>false</code>.

Syntax	<code><bool> within (<string> geoJSONGeometry , <string> geoJSONGeometryFence)</code>
Extension Type	Function

Description	Returns true if the geoJSONGeometry is within the geoJSONGeometryFence. Returns false otherwise.
Example	<ul style="list-style-type: none"> 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.

withindistance

Syntax	<code><bool> withindistance (<double> longitude , <double> latitude, <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns true if the location specified in terms of longitude and latitude is within distance of the geoJSONGeometryFence. Returns false otherwise.
Example	<code>withindistance(0.5 , 0.5 , {'type':'Polygon','coordinates':[[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]]} , 110574.61087757687)</code> returns true because the location specified in terms of longitude and latitude is within the distance of the geoJSONGeometryFence.

Syntax	<code><bool> withindistance (<string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> distance)</code>
Extension Type	Function
Description	Returns true if the area given by geoJSONGeometry is within distance of the geoJSONGeometryFence.
Example	<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]]]} , 110574.61087757687)</code> returns true because geoJSONGeometry is within the distance of geoJSONGeometryFence.

crosses

Syntax	<code><bool> crosses (<string> id , <double> longitude, <double> latitude , <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns true when the the specified object of which the location is specified in terms of longitude and latitude crosses the geographic location specified in geoJSONGeometryFence . Returns false when the object crosses out of the location specified in geoJSONGeometryFence .

Example	<ul style="list-style-type: none"> • <code>crosses(km-4354, -0.5, 0.5, {'type':'Polygon','coordinates':[[[0, 0],[2, 0],[2, 1],[0, 1],[0, 0]]})</code> returns true. • <code>km-4354, 1.5, 0.5, {'type':'Polygon','coordinates':[[[0, 0],[2, 0],[2, 1],[0, 1],[0, 0]]})</code> returns true.
----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Syntax	<code><bool> crosses (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence)</code>
Extension Type	StreamProcessor
Description	Returns true when the object (i.e. <code>geoJSONGeometry</code>) crosses the specified geographic location (i.e. <code>geoJSONGeometryFence</code>). Returns false when the object crosses out of <code>geoJSONGeometryFence</code> .

stationary

Syntax	<code><bool> stationary (<string> id , <double> longitude, <double> latitude , <string> geoJSONGeometryFence , <double> radius)</code>
Extension Type	StreamProcessor
Description	Returns true when the object (defined in terms of <code>longitude</code> and <code>latitude</code>) becomes stationary within the specified <code>radius</code> . Returns false when the object moves out of the specified <code>radius</code> .
Example	<ul style="list-style-type: none"> • <code>stationary(km-4354,0,0, 110574.61087757687)</code> returns true. • <code>stationary(km-4354,1,1, 110574.61087757687)</code> returns true . • <code>stationary(km-4354,1,1.5, 110574.61087757687)</code> returns true.

Syntax	<code><bool> stationary (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> radius)</code>
Extension Type	StreamProcessor
Description	Returns true when the object (i.e. <code>geoJSONGeometry</code>) becomes stationary within the specified <code>radius</code> . Returns false when the objects moves out of the specified radius.

proximity

Syntax	<code><bool,string> proximity (<string> id , <double> longitude, <double> latitude , <string> geoJSONGeometryFence , <double> radius)</code>
Extension Type	StreamProcessor
Description	Returns true when two objects (specified in terms of <code>longitude</code> and <code>latitude</code>) are within the specified <code>radius</code> to another object. Returns false when the specified object moves out of the specified <code>radius</code> . The <code>proximityWith</code> optional attribute indicates the ID of the object that the object specified is in close proximity with. <code>proximityID</code> is a unique ID for the two objects in close proximity.

Example	The following return true with ID3. <ul style="list-style-type: none"> • 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	<code><bool> proximity (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> radius)</code>
Extension Type	StreamProcessor
Description	Returns <code>true</code> when an object (i.e. <code>geoJSONGeometry</code>) is within the specified <code>radius</code> from another object. Returns <code>false</code> when one or both objects move away from each other and are no longer within the specified <code>radius</code> s of each other. The <code>proximityWith</code> optional attribute indicates the ID of the object that the object specified is in close proximity with. <code>proximityID</code> is a unique ID for the two objects in close proximity.

geocode

Syntax	<code><double, double, string> geocode (<string> location)</code>
Extension Type	StreamProcessor
Description	Transforms a location to its geo-coordinates (<code>longitude</code> and <code>latitude</code>) and formatted address.
Example	geocode(duplication rd) returns the following data with adhering latitude, longitude, and formattedAddress attribute names respectively. 6.8995244d, 79.8556202d, "R A De Mel Mawatha, Colombo, Sri Lanka"

r

This extension allows you to execute R scripts within Siddhi query. Following are the functions of the R extension.

eval

Syntax	<code>[<int long float double string boolean> output1 , <int long float double string boolean> output2, ...] eval (<string > script, <string > outputAttributes, <int long float double string boolean > input1 , <int long float double string boolean> input2 , ...)</code>
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	<code>script</code> : R script as a string produces aggregated outputs based on the provided input variable parameters and expected output attributes. <code>outputAttributes</code> : A set of output attributes separated by commas as string. Each attribute is denoted as <code><name><space><type></code> . e.g., 'output1 string, output2 long'
Return Parameters	Output parameters are generated based on the <code>outputAttributes</code> 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.
----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

evalSource

Syntax	[<int long float double string boolean> output1 , <int long float double string boolean> output2 , ...] eval (<string> filePath , <string> outputAttributes , <int long float double string boolean> input1 , <int long float double string boolean> 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	eval('/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.

regex

This extension provides basic RegEx execution capabilities to Siddhi. Following are the functions of the RegEx extension.

find

Syntax	<bool> find (<string> regex , <string> inputSequence)
Extension Type	Function
Description	This method attempts to find the next sub-sequence of the inputSequence that matches the regex pattern. It returns true if such a sub sequence exists, or returns false otherwise.
Examples	<ul style="list-style-type: none"> • <code>find("\d\d(.*)WSO2", "21 products are produced by WSO2 currently")</code> returns true. • <code>find("\d\d(.*)WSO2", "21 products are produced currently")</code> returns false.

Syntax	<bool> find (<string> regex , <string> inputSequence , <int> startIndex)
Extension Type	Function

Description	This method attempts to find the next sub-sequence of the inputSequence that matches the regex pattern starting from given index (i.e. startingIndex). It returns true if such a subsequence exists, or returns false otherwise.
Examples	<ul style="list-style-type: none"> • <code>find("\d\d(.*)WSO2", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 30)</code> returns true. • <code>find("\d\d(.*)WSO2", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 35)</code> returns false.

group

Syntax	<code><string> group (<string> regex , <string> inputSequence , <int> groupId)</code>
Extension Type	Function
Description	Returns the input sub-sequence captured by the given group during the previous match operation. Returns null if no sub-sequence was found during the previous match operation. For more information about the match operation, see matches .
Example	<code>group("(\d\d)(.*)(WSO2.*)", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 3)</code> returns "WSO2 employees".

lookingAt

Syntax	<code><string> lookingAt (<string> regex , <string> inputSequence)</code>
Extension Type	Function
Description	This method attempts to match the inputSequence against the regex pattern starting at the beginning.
Examples	<ul style="list-style-type: none"> • <code>lookingAt("\d\d(.*)WSO2", "21 products are produced by WSO2 currently in Sri Lanka")</code> returns true. • <code>lookingAt("WSO2(.*)middleware(.*)", "sample test string and WSO2 is situated in trace and its a middleware company")</code> returns false.

matches

Syntax	<code><string> matches (<string> regex , <string> inputSequence)</code>
Extension Type	Function
Description	This method attempts to match the entire inputSequence against the regex pattern.
Examples	<ul style="list-style-type: none"> • <code>matches("WSO2(.*)middleware(.*)", "WSO2 is situated in trace and its a middleware company")</code> returns true. • <code>matches("WSO2(.*)middleware", "WSO2 is situated in trace and its a middleware company")</code> returns false.

time

This extension provides time related functionality to Siddhi such as getting current time, current date, manipulating/formatting dates, etc. Following are the functions of the time extension.

currentDate

Syntax	<string> currentDate ()
Extension Type	Function
Description	Returns the current system date in the yyyy-MM-dd format.
Example	currentDate() returns 2015-08-20.

currentTime

Syntax	<string> currentTime ()
Extension Type	Function
Description	Returns the current system time in the HH:mm:ss format.
Example	currentTime() returns 13:15:10.

currentTimestamp

Syntax	<string> currentTimestamp ()
Extension Type	Function
Description	Returns the current system timestamp in the yyyy-MM-dd HH:mm:ss format.
Example	currentTimestamp() returns 2015-08-20 13:15:10.

dateAdd

The common parameters of this function are described below.

- **dateValue** : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- **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> 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	dateAdd("2014-11-11 13:23:44", 2, 'year', "yyyy-MM-dd HH:mm:ss") returns "2016-11-11 13:23:44".

Syntax	<string> 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	dateAdd("2014-11-11 13:23:44", 2, 'year') returns "2016-11-11 13:23:44".

Syntax	<string> 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".

dateSub

The common parameters of this function are described below.

- **dateValue** : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- **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> 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	dateSub("2014-11-11 13:23:44", 2, 'year', "yyyy-MM-dd HH:mm:ss") returns "2012-11-11 13:23:44".

Syntax	<string> dateSub (<string> dateValue , < long > expr , <string> unit)
Extension Type	Function
Description	Returns the specified date and time stamp with the selected unit of the specified dateValue reduced by the given amount (i.e. expr).
Example	dateSub("2014-11-11 13:23:44", 2, 'year') returns "2012-11-11 13:23:44".

Syntax	<string> dateSub (<long> timestampInMilliseconds , < long > expr , <string> unit)
Extension Type	Function

Description	Returns the specified time stamp with the selected unit of the specified timestampInMilliseconds reduced by the given amount (i.e. expr).
Example	<code>dateSub(1415692424000L, 2, 'year')</code> returns 1352620424000.

dateDiff

The common parameters of this function are described below.

- **dateValue1** : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- **dateValue2** : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- **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	<code><int> dateDiff (<string> dateValue1 , < string > dateValue2 , <string> dateFormat1, <string> dateFormat2)</code>
Extension Type	Function
Description	Returns the number of days between the two dates specified (i.e. dateValue1 and dateValue2).
Example	<code>dateDiff('2014-11-11 13:23:44', '2014-11-9 13:23:44', 'yyyy-MM-dd HH:mm:ss', 'yyyy-MM-dd HH:mm:ss')</code> returns 2.

Syntax	<code><int> dateDiff (<string> dateValue1 , < string > dateValue2)</code>
Extension Type	Function
Description	Returns the number of days between the two dates specified (i.e. dateValue1 and dateValue2).
Example	<code>dateDiff('2014-11-11 13:23:44.000', '2014-11-9 13:23:44.000')</code> returns 2.

Syntax	<code><int> dateDiff (<string> timestampInMilliseconds1 , < string > timestampInMilliseconds2)</code>
Extension Type	Function
Description	Returns the number of days between the two date and time stamps specified (i.e. timestampInMilliseconds1 and timestampInMilliseconds2).
Example	<code>dateDiff(1415692424000, 1415519624000)</code> returns 2.

dateFormat

The common parameters of this function are described below.

- **dateValue** : -A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- **dateTargetFormat** : The date format to which the specified **date value** needs to be converted. e.g., yyyy/MM/dd HH:mm:ss
- **dataSourceFormat** : 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	<string> dateFormat (<string> dateValue ,<string> dateTargetFormat ,<string> dataSourceFormat)
Extension Type	Function
Description	Returns a formatted date string.
Example	<code>dateFormat('2014-11-11 13:23:55', 'ss', 'yyyy-MM-dd HH:mm:ss')</code> returns 55.

Syntax	<string> dateFormat (<string> dateValue ,<string> dateTargetFormat)
Extension Type	Function
Description	Returns a formatted date string.
Example	<code>dateFormat('2014-11-11 13:23:55.657', 'ss')</code> returns 55.

Syntax	<string> dateFormat (<long> timestampInMilliseconds ,<string> dateTargetFormat)
Extension Type	Function
Description	Returns a formatted date string.
Example	<code>dateFormat(1415692424000, 'yyyy-MM-dd')</code> returns 2014-11-11.

extract

- **dateValue** : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- **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** : A date value in milliseconds (from the epoch) e.g., 1415712224000L

Syntax	<int> extract (<string> unit ,<string> dateValue , <string> dateFormat)
Extension Type	Function
Description	Returns the specified unit extracted from the specified dateValue .
Example	<code>extract('year', '2014-3-11 02:23:44', 'yyyy-MM-dd hh:mm:ss')</code> returns 2014.

Syntax	<int> extract (<string> unit ,<string> dateValue)
Extension Type	Function
Description	Returns the specified unit extracted from the specified dateValue .
Example	<code>extract('year', '2014-3-11 02:23:44.234')</code> returns 2014.

Syntax	<code><int> extract (<long> timestampInMilliseconds ,<string> unit)</code>
Extension Type	Function
Description	Returns the specified <code>unit</code> extracted from the specified <code>timestampInMilliseconds</code> .
Example	<code>extract(1394484824000, 'year')</code> returns 2014.

date

Syntax	<code><string> date (<string> dateValue ,<string> dateFormat)</code>
Extension Type	Function
Description	Returns the date component of the <code>dateValue</code> .
Example	<code>extract('2014-11-11 13:23:44', 'yyyy-MM-dd HH:mm:ss')</code> returns 2014-11-11.

timestampInMilliseconds

Syntax	<code><long> timestampInMilliseconds ()</code>
Extension Type	Function
Description	Returns the current time stamp in milliseconds.
Example	<code>timestampInMilliseconds()</code> returns 1440160328693.

Syntax	<code><long> timestampInMilliseconds (<string> dateValue)</code>
Extension Type	Function
Description	Returns the time stamp of the specified <code>dateValue</code> in milliseconds. In order to use this function, the date format of the specified <code>dateValue</code> should be <code>yyyy-MM-dd HH:mm:ss.SSS</code> .
Example	<code>timestampInMilliseconds('2007-11-30 10:30:19.000')</code> returns 1196398819000.

Syntax	<code><long> timestampInMilliseconds (<string> dateValue, <string> dateFormat)</code>
Extension Type	Function
Description	Returns the time stamp of the specified <code>dateValue</code> in milliseconds. The date format can be specified in the <code>dateFormat</code> parameter.
Example	<code>timestampInMilliseconds('2007-11-30 10:30:19', 'yyyy-MM-dd HH:mm:ss')</code> returns 1196398819000.

utcTimestamp

Syntax	<code><string> utcTimestamp()</code>
Extension Type	Function
Description	Returns the system time in the <code>yyyy-MM-dd HH:mm:ss</code> date format.

Example	utcTimestamp() returns 2015-08-21 12:16:13.
----------------	---------------------------------------------

nlp

This extension provides Natural Language Processing capabilities to Siddhi. Functions of the NLP extension are as follows.

findNameEntityType

Syntax	<code><string> findNameEntityType(<string> entityType, <bool> groupSuccessiveMatch, <string> string-variable)</code>
Extension Type	Function
Description	<p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> entityType : This is a user-specified string constant. e.g., PERSON, LOCATION, ORGANIZATION, MONEY, PERCENT, DATE or TIME groupSuccessiveMatch : This is a user-specified boolean constant used to group successive matches of the specified entityType and a text stream. streamAttribute : A string or the stream attribute in which text stream is included. <p>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.</p>
Example	<pre>findNameEntityType("PERSON", true, text)</pre> <p>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.</p>

findNameEntityTypeViaDictionary

Syntax	<code><string> findNameEntityTypeViaDictionary(<string> entityType, <string> dictionaryFilePath, <string> string-variable)</code>
Extension Type	Function
Description	<p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> 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. streamAttribute : A string or the stream attribute in which text stream is included. <p>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.</p>
Example	<pre>findNameEntityTypeViaDictionary("PERSON", "dictionary.xml",text)</pre> <p>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".</p>

findRelationshipByVerb

Syntax	<code><string > text, <string> subject, < string > object < string > verb findRelationshipByVerb (<string> verb, <string> string-variable)</code>
Extension Type	Function
Description	<p><code>findRelationshipByVerb</code> 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.</p> <p>The input parameters used are as follows.</p> <ul style="list-style-type: none"> • verb : This is a user specified string constant. • string-variable : A string or the stream attribute which includes the text stream.
Examples	<pre>findRelationshipByVerb("say", "Information just reaching us says another Liberian With Ebola Arrested At Lagos Airport") returns the following.</pre> <ul style="list-style-type: none"> • The whole text • Information as the subject • Liberian as the object. • says as the verb.

findRelationshipByRegex

Syntax	<code><string > text, <string> subject, < string > object < string > verb findRelationshipByRegex (<string> regex, <string> string-variable)</code>
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	<pre>findRelationshipByRegex(' { }=verb /nsubj agent/ { }=subject /dobj/ { }=object', "gates foundation donates \$50M in support of #Ebola relief") returns the following.</pre> <ul style="list-style-type: none"> • The whole text • "foundation" as the subject • "\$" as the object • "donates" as the verb

findSemgrexPattern

Syntax	<code><string > text, <string> match, < string > object < string > verb findSemgrexPattern (<string> regex, <string> string-variable)</code>
Extension Type	Function
Description	<p>The <code>findSemgrexPattern</code> function returns the whole text, subject, object and verb from the text stream that matches the named nodes of the Semgrex pattern.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • 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	<pre>findSemgrepPattern('{lemma:die} >.*subj num.*=/reln {}=diedsubject', "Sierra Leone doctor dies of Ebola after failed evacuation.")</pre> <p>In this example, the function searches for words with the lemmatization <code>die</code> that are governors on any subject or numeric relation. The dependent is marked as the <code>diedsubject</code>, and the relationship is marked as <code>reln</code>. Thus, the query returns an output stream that has the full match of this expression, i.e. the governing word with lemmatization for <code>die</code>. It also returns the name of the corresponding node for each match it finds.</p> <p>The following is the list of elements in the output stream.</p> <ul style="list-style-type: none"> • The whole text • <code>dies</code> as the match • "<code>nsubj</code>" as <code>reln</code> • <code>doctor</code> as <code>diedsubject</code>
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

findTokensRegexPattern

Syntax	<pre>< string > text, <string> match, <string> group_1, etc. findTokensRegexPattern (<string> regex, <string> string-variable)</pre>
Extension Type	Function
Description	<p><code>findTokensRegexPattern</code> returns the whole text, subject, object and verb from the text stream that matches the named nodes of the Semgrep pattern. The return also includes the corresponding node in the Semgrep pattern and the corresponding named relation defined in the regular expression for each word/phrase.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • regex : A user specified regular expression that matches the Semgrep pattern syntax. • string-variable : A string or the stream attribute which includes the text stream.
Example	<p><code>findTokensRegexPattern('([ner:/PERSON ORGANIZATION LOCATION/]*) (?:[]*[lemma:donate]) ([ner: MONEY]+)', text)</code> defines three groups:</p> <ul style="list-style-type: none"> • 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. <p>This function returns the following.</p> <ul style="list-style-type: none"> • The whole text • " Paul Allen donates \$ 9million " as the match. • " Paul Allen", as <code>group_1</code>. • "\$ 9million" as <code>group_2</code>.

pmmI

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 PMML extension are as follows.

predict

Syntax	<pre>< double float long int string boolean > predict(<string> pathToPmmlFile)</pre>
---------------	-----------------------------------------------------------------------------------------------------------

Extension Type	Stream Processor
Description	<p>Processes the input stream attributes according to the defined PMML standard model and outputs the processed results together with the input stream attributes.</p> <p>This function uses the following input parameter.</p> <ul style="list-style-type: none"> • pathToPmmlFile : The path to the PMML model file. <p>The function returns the outputs defined in the output fields. The number of outputs can vary.</p>
Example	<pre>predict('<DAS HOME>/samples/artifacts/0301/decision-tree.pmml')</pre> <p>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.</p>

Syntax	<code>< double float long int string boolean > predict(<string> pathToPmmlFile, <double float long int string boolean> input)</code>
Extension Type	Stream Processor
Description	<p>Processes the input stream attributes according to the defined PMML standards model and outputs the processed results.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • 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 <code>predict</code> function does not accept any constant values as input parameters. You can have multiple input parameters according to the input stream definition. <p>This function returns the processed outputs defined in the query. The number of outputs can vary depending on the query definition.</p>
Examples	<pre>predict('<DAS_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)</pre> <p>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.</p>

ml

This extension provides Siddhi the capability to make predictions based on Machine Learning models. Supported functions of the ML extension are as follows.

`predict`

Syntax	<code><double float long int string boolean> predict(<string> pathToMLModel, <string> dataType)</code>
---------------	--------------------------------------------------------------------------------------------------------------------------

Extension Type	Stream Processor
Description	<p>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.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • pathToMLModel : The file path or the registry path to the location of the ML model. If the model storage location is <code>registry</code>, the value of this parameter should have <code>registry</code> as the prefix. • dataType : The data type of the predicted value (double, float, long, integer/int, string, boolean/bool).
Example	<code>predict('registry:_system/governance/mlmodels/indian-diabetes-model')</code>

Syntax	<code><double float long int string boolean> predict(<string> pathToMLModel, <stype, <double> input)</code>
Extension Type	Stream Processor
Description	<p>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.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • pathToMLModel : The file path or the registry path to the location of the ML model. If the model storage location is <code>registry</code>, the value of this parameter should have <code>registry</code> 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 prediction is made. The <code>predict</code> function does not accept any constant values as input parameters. You can send multiple input parameters.
Example	<code>predict('registry:_system/governance/mlmodels/indian-diabetes-model', NumTSFT, DPF, BMI, DBP, PG2, Age, SI2)</code>

timeseries

See [Time Series Extension](#).

kf (Kalman Filter)

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

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 in this function 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.

Syntax	<code><double, double> kf:kalmanFilter(<double> measuredValue)</code>
Extension Type	Function
Example	<ul style="list-style-type: none"> 1st round: <code>kf:kalmanFilter(-74.178444)</code> returns an estimated value of <code>-74.178444</code>. 2nd round: <code>kf:kalmanFilter(-74.175703)</code> returns an estimated value of <code>-74.1770735006853</code>. 3rd round: <code>kf:kalmanFilter(-74.177872)</code> returns an estimated value of <code>-74.1773396670348</code>.

Syntax	<code><double, double> kf: kalmanFilter(<double> measuredValue, <double> measurementNoiseSD)</code>
Extension Type	Function
Example	<ul style="list-style-type: none"> 1st round: <code>kf:kalmanFilter(-74.178444, 0.003)</code> returns an estimated value of <code>-74.178444</code>. 2nd round: <code>kf:kalmanFilter(-74.175703, 0.003)</code> returns an estimated value of <code>-74.17707350205573</code>. 3rd round: <code>kf:kalmanFilter(-74.177872, 0.003)</code> returns an estimated value of <code>-74.177339667771</code>.

Syntax	<code><double, double> kf: kalmanFilter(<double> measuredValue, <double> measuredChangingRate, <double> measurementNoiseSD, <long> timestamp)</code>
Extension Type	Function
Example	<ul style="list-style-type: none"> 1st round: <code>kf:kalmanFilter(-74.178444, 0.003, 0.01, time:timestampInMilliseconds())</code> returns an estimated value of <code>-74.1784439700006</code>. 2nd round: <code>kf:kalmanFilter(-74.178444, 0.003, 0.01, time:timestampInMilliseconds())</code> returns an estimated value of <code>-74.1784439700006</code>. 3rd round: <code>kf:kalmanFilter(-74.177872, 0.003, 0.01, time:timestampInMilliseconds())</code> returns an estimated value of <code>-74.17697314316393</code>.

map

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

Syntax	<code><Object> create()</code> or <code><Object> create(<Object> key1, <Object> value1, <Object> key2, <Object> value2, ... <Object> keyN, <Object> valueN)</code>
Extension Type	Function
Description	Returns the created map object.

Examples	<ul style="list-style-type: none"> • <code>create()</code> returns an empty map. • <code>create(1, "one", 2, "two", 3, "three")</code> returns a map with keys 1, 2, 3 and corresponding values "one", "two", "three".
-----------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

get

Syntax	<code><Object> get(<Map> map, <Object> key)</code>
Extension Type	Function
Description	Returns the value object from the map that is related to the given key.
Example	<code>get(company, 1)</code> returns the value that is related to the key 1 from the map named company .

isMap

Syntax	<code><bool> isMap(<Object> object)</code>
Extension Type	Function
Description	Returns <code>true</code> if the object is a map or <code>false</code> otherwise.
Example	<code>isMap(students)</code> returns <code>true</code> if the <code>students</code> object is a map. It returns <code>false</code> if the <code>students</code> object is not a map.

put

Syntax	<code>Object> put(<Object> map, <Object> key, <Object> value)</code>
Extension Type	Function
Description	Returns the updated map after adding the given key-value pair.
Example	<code>put(students, 1234, "sam")</code> returns the updated map named <code>students</code> after adding the object "sam" with key 1234.

remove

Syntax	<code><Object> remove(<Object> map, <Object> key)</code>
Extension Type	Function
Description	Returns the updated map after removing the element with key.
Example	<code>remove(students, 1234)</code> returns the updated map <code>students</code> after removing the element with the key 1234.

createFromJSON

Syntax	<code><Object> createFromJSON(<string> JSONstring)</code>
Extension Type	Function

Description	Returns the map created with the key values pairs given in the JSONstring.
Example	createFromJSON("{'symbol' : 'IBM', 'price' : 200, 'volume' : 100}") returns a map with the keys "symbol", "price", "volume", and with the values "IBM", 200 and 100 respectively.

createFromXML

Syntax	<Object> createFromXML(<string> XMLstring)
Extension Type	Function
Description	Returns the map created with the key values pairs given in the XMLstring.
Example	createFromXML("<> <company> <symbol> wso2 </symbol> <price> <100> </price> <volume> 200 </volume> </company>") returns a map with the keys "symbol", "price", "volume", and with the values WSO2, 100 and 200 respectively.

toJSON

Syntax	<string> toJSON(<Object> map)
Extension Type	Function
Description	Converts a map into a JSON object and returns the definition of that JSON object as a string.
Example	If "company" is a map with key value pairs ("symbol" : wso2), ("volume" : 100), and ("price", 200), toJSON(company) returns the string "{"symbol" : "wso2", "volume" : 100, "price" : 200}".

toXML

Syntax	<string> toXML(<Object> map)
Extension Type	Function
Description	Returns the map as an XML string.
Example	If "company" is a map with key value pairs ("symbol" : wso2), ("volume" : 100), and ("price" : 200), toXML(map) returns the string "<symbol>wso2</symbol><volume><100></volume><price>200</price>".

reorder

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.

```
@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.
kslack

Syntax	<bool> kslack(<long> timestamp)
Extension Type	StreamProcessor
Description	This is the most basic version. The events are sorted by the <code>timestamp</code> parameter.

Syntax	<bool> kslack(<long> timestamp, <long> timeOut)
Extension Type	StreamProcessor
Description	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.

Syntax	<bool> kslack(<long> timestamp, <long> timeOut, <long> maxValue)
Extension Type	StreamProcessor
Description	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.
Example	<ul style="list-style-type: none"> • <code>kslack(timestamp, -11, 1000000)</code> 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 -11 which effectively disables the timer based draining of the internal buffer. • <code>kslack(timestamp, 10001, 1000000)</code> 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	<bool> kslack(<long> timestamp, <long> timeOut, <bool> expireFlag)
Extension Type	StreamProcessor
Description	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.
Example	<code>kslack(timestamp, -11, -11, true)</code>

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

- Forecast

- Outlier
- Regression
- kernelMinMax (Kernel based minima maxima detection)
- kalmanMinMax (kalman 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.

Parameter	Description	Required/Optional	Default Value
Calculation Interval	The frequency with which the regression calculation should be carried out.	Optional	1 (i.e., for every event)
Batch Size	The maximum number of events that should be used for a regression calculation.	Optional	1,000,000,000
Confidence Interval	The confidence interval to be used for a regression calculation.	Optional	0.95
Next X Value	The value to be used to forecast the Y value. This can be a constant or an expression (e.g., x+5).	Required	
Y Stream	The data stream of the dependent variable.	Required	
X Stream	The data stream of the independent variable.	Required	

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.

Parameter	Description	Required/Optional	Default Value
Time Window	The maximum time duration that should be considered for a regression calculation.	Required	
Batch Size	The maximum number of events that shoukd be used for a regression calculation.	Required	
Next X Value	The value to be used to forecast the Y value. This can be a constant or an expression (e.g., x+5).	Required	
Calculation Interval	The frequency with which the regression calculation should be carried out.	Optional	1 (i.e., for every event)

Confidence Interval	The confidence interval to be used for a regression calculation.	Optional	0 . 95
Y Stream	The data stream of the dependent variable.	Required	
X Stream	The data stream of the independent variable.	Required	

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.

Parameter	Name	Description
Forecast Y	forecastY	The forecast Y value based on next X and regression equation.
Standard Error	stdError	The standard error of the regression equation.
coefficients	beta0, beta1	coefficients of the simple linear regression.
Input Stream Data	The name given in the input stream.	All the items sent in the input stream.

Examples

The queries given in the examples below return the following wen 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.

```
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.

```
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 in 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.

Parameter	Description	Required/Optional	Default Value
Calculation Interval	The frequency with which the regression calculations should be carried out.	Optional	1 (i.e., for every event)
Batch Size	The maximum number of events to be used for a regression calculation.	Optional	100,000,000
Confidence Interval	The confidence interval to be used for a regression calculation.	Optional	0.95
Range	The number of standard deviations from the regression equation.	Required	0.95
Y Stream	The data stream of the dependent variable.	Required	
X Stream	The data stream of the independent variable.	Required	

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.

Parameter	Description	Required/Option	Default Value
Time Window	The maximum time duration to be considered for a regression calculation.	Required	
Batch Size	The maximum number of events to be used for a regression calculation.	Required	
Range	The number of standard deviations from the regression calculation.	Required	
Calculation Interval	The frequency with which the regression calculation should be carried out.	Optional	1 (for every event)
Confidence Level	The confidence interval to be used for a regression calculation.	Optional	0.95
Y Stream	The data stream of the dependent variable.	Required	

X Stream	The data stream of the independent variable.	Required	
----------	----------------------------------------------	----------	--

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.

The same output parameters are available for each implementation.

Parameter	Name	Description
Outlier	outlier	True if the event is an outlier, False if not.
Standard Error	stdError	The standard error of the regression equation.
coefficients	beta0, beta1	coefficients of the regression equation.
Input Stream Data	The name given in the input stream.	All the items sent in the input stream.

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 (), 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.

```
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.

```
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

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \varepsilon$$

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.

Parameter	Description	Required/Optional	Default Value
Calculation Interval	The frequency with which the regression calculation should be carried out.	Optional	1 (i.e., for every event)
Batch Size	The maximum number of events to be used for a regression calculation.	Optional	1,000,000,000
Confidence Interval	The confidence interval to be used for a regression calculation.	Optional	0.95
Y Stream	The data stream of the dependent variable.	Required	
X Stream(s)	The data stream(s) of the independent variable.	Required	

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.

Parameter	Description	Required/Optional	Default Value
Time Window	The maximum time duration to be considered for the regression calculation.	Required	
Batch Size	The maximum number of events to be used for a regression calculation.	Required	
Calculation Interval	The frequency with which the regression calculation should be carried out.	Optional	1 (for every event)
Confidence Interval	The confidence interval to be used for a regression calculation.	Optional	0.95
Y Stream	The data stream of the dependent variable.	Required	
X Stream(s)	The data stream(s) of the independent variable.	Required	

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.

Parameter	Name	Description

Standard Error	stdError	The standard error of the regression equation.
coefficients	beta0, beta1, beta2 etc.	n+1 coefficients where n is the number of x parameter s.
Input Stream Data	The name given in the input stream	All the attributes sent in the input stream.

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.

```
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 ($\beta_0, \beta_1, \beta_2, \beta_3$) 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.

```
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 (β_0, β_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

Parameter	Name	Description
min or max value	The variable name specified in the input parameter.	The value of the min or max point.
extremaType	extrema Type	Indicates whether the returned value is a min value or a max value.

Examples

The following returns the maximum values for a set of price values.

```
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.

```
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.

```
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

Parameter	Required/Optional	Description
Variable	Required	The time series value to be considered for minima maxima detection.
Q	Required	The standard deviation of the process noise.
R	Required	The standard deviation of the measurement noise.
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

Parameter	Name	Description
min or max value	The variable name specified in the input parameter.	The value of the min or max point.
extrema Type	extrema Type	Indicates whether the returned value is a min value or a max value.

Examples

The following returns the maximum values for a set of `price` values.

```
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.

```
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
- In 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

Syntax	<code><double> math:abs(<float double> p1)</code>
Extension Type	Function
Description	Returns the absolute value of <code>p1</code> . This function wraps the <code>java.lang.Math.abs()</code> function.
Examples	Both the following queries return 3 since the absolute value of both 3 and -3 is 3. <ul style="list-style-type: none"> • <code>abs(3)</code> • <code>abs(-3)</code>

acos function

Syntax	<code><double> math:acos(<float double> p1)</code>
Extension Type	Function
Description	If $-1 \leq p1 \leq 1$, this function returns the arc-cosine (inverse cosine) of <code>p1</code> . If not, it returns <code>NULL</code> . The return value is in radian scale. This function wraps the <code>java.lang.Math.acos()</code> function.
Example	<code>acos(0.5)</code> returns <code>1.0471975511965979</code> .

asin function

Syntax	<code><double> math: asin (<float double> p1)</code>
Extension Type	Function
Description	If $-1 \leq p1 \leq 1$, this function returns the arc-sin (inverse sine) of <code>p1</code> . If not, it returns NULL. The return value is in radian scale. This function wraps the <code>java.lang.Math.asin()</code> function.
Example	<code>asin(0.5)</code> returns 0.5235987755982989.

atan function

Syntax	<code><double> math: atan(<int long float double> p1)</code>
Extension Type	Function
Description	Returns the arc-tangent (inverse tangent) of <code>p1</code> . The return value is in radian scale. This function wraps the <code>java.lang.Math.atan()</code> function.
Examples	<code>atan(6d)</code> returns 1.4056476493802699.

Syntax	<code><double> math: atan (<int long float double> p1, <int long float double> p2)</code>
Extension Type	Function
Description	Returns the arc-tangent (inverse tangent) of <code>p1</code> and <code>p2</code> coordinates. The return value is in radian scale. This function wraps the <code>java.lang.Math.atan2()</code> function.
Examples	<code>atan(12d, 5d)</code> returns 1.1760052070951352.

Binary function

Syntax	<code><string> math: bin(<int long> p1)</code>
Extension Type	Function
Description	Returns a string representation of the integer/long <code>p1</code> argument as an unsigned integer in base 2. This function wraps the <code>java.lang.Integer.toBinaryString</code> and <code>java.lang.Long.toBinaryString</code> methods.
Examples	<code>bin(9)</code> returns "1001".

Ceiling function

Syntax	<code><double> math: ceil(<float double> p1)</code>
Extension Type	Function

Description	Returns the smallest (closest to negative infinity) double value that is greater than or equal to the <code>p1</code> argument, and is equal to a mathematical integer. This function wraps the <code>java.lang.Math.ceil()</code> method.
Example	<code>ceil(423.187d)</code> returns <code>424.0</code> .

Convert function

Syntax	<code><string> math: conv(<string> a, <int> fromBase, <int> toBase)</code>
Extension Type	Function
Description	Converts <code>a</code> from the <code>fromBase</code> base to the <code>toBase</code> base.
Example	<code>conv("7f", 16, 10)</code> returns <code>"127"</code> .

CopySign function

Syntax	<code><double> math: copySign(<int long float double> magnitude, <int long float double> sign)</code>
Extension Type	Function
Description	Returns the magnitude of <code>magnitude</code> with the sign of <code>sign</code> . This function wraps the <code>java.lang.Math.copySign()</code> function.
Example	<code>copySign(5.6d, -3.0d)</code> returns <code>-5.6</code> .

cos function

Syntax	<code><double> math: cos(<int long float double> p1)</code>
Extension Type	Function
Description	Returns the cosine of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.cos()</code> function.
Example	<code>cos(6d)</code> returns <code>0.9601702866503661</code> .

cosh function

Syntax	<code><double> math: cosh(<int long float double> p1)</code>
Extension Type	Function
Description	Returns the hyperbolic cosine of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.cosh()</code> function.
Example	<code>cosh(6d)</code> returns <code>201.7156361224559</code> .

Cube Root function

Syntax	<code><double> math: cbrt(<int long float double> p1)</code>
Extension Type	Function
Description	Returns the cube-root of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.cbrt()</code> function.
Example	<code>cbrt(17d)</code> returns <code>2.5712815906582356</code> .

e function

Syntax	<code><double> math: e()</code>
Extension Type	Function
Description	Returns the <code>java.lang.Math.E</code> constant, which is the closest double value to <code>e</code> (which is the base of the natural logarithms).
Example	<code>e()</code> returns <code>2.7182818284590452354</code> .

Exponential function

Syntax	<code><double> math: exp(<int long float double> p1)</code>
Extension Type	Function
Description	Returns Euler's number <code>e</code> raised to the power of <code>p1</code> . This function wraps the <code>java.lang.Math.exp()</code> function.
Example	<code>exp(10.23)</code> returns <code>27722.51006805505</code> .

Floor function

Syntax	<code><double> math: floor(<int long float double> p1)</code>
Extension Type	Function
Description	This function wraps the <code>java.lang.Math.floor()</code> function that returns the largest (closest to positive infinity) value that is less than or equal to <code>p1</code> , and is equal to a mathematical integer.
Example	<code>floor(10.23)</code> returns <code>10.0</code> .

Get Exponent function

Syntax	<code><double> math: getExponent(<int long float double> p1)</code>
Extension Type	Function

Description	Returns the unbiased exponent used in the representation of <code>p1</code> . This function wraps the <code>java.lang.Math.getExponent()</code> function.
Example	<code>getExponent(60984.1)</code> returns 15.

Hexadecimal function

Syntax	<code><string> math: hex(<int long float double> p1)</code>
Extension Type	Function
Description	This function wraps the <code>java.lang.Double.toHexString()</code> function that returns a hexadecimal string representation of <code>p1</code> .
Example	<code>hex(200)</code> returns "c8".

Is Infinite function

Syntax	<code><boolean> math: isInfinite(<float double> p1)</code>
Extension Type	Function
Description	This function wraps the <code>java.lang.Float.isInfinite()</code> and <code>java.lang.Double.isInfinite()</code> functions that return <code>true</code> if <code>p1</code> is infinitely large in magnitude, or return <code>false</code> otherwise.
Example	<code>isInfinite(java.lang.Double.POSITIVE_INFINITY)</code> returns <code>true</code> .

Is Not A Number function

Syntax	<code>< boolean> math: isNaN(<float double> p1)</code>
Extension Type	Function
Description	This function wraps the <code>java.lang.Float.isNaN()</code> and <code>java.lang.Double.isNaN()</code> functions that return <code>true</code> if <code>p1</code> is a NaN (Not-a-Number) value, or return <code>false</code> otherwise.
Example	<code>isNaN(java.lang.Math.log(-12d))</code> returns <code>true</code> .

In function

Syntax	<code><double> math: ln (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the natural logarithm (base e) of <code>p1</code> .
Example	<code>ln(11.453)</code> returns 2.438251704415579.

log2 function

Syntax	<code><double> math: log2 (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the base 2 logarithm of <code>p1</code> .
Example	<code>log2(91d)</code> returns 6.507794640198696.

log10 function

Syntax	<code><double> math: log10 (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the base 10 logarithm of <code>p1</code> .
Example	<code>log10(19.234)</code> returns 1.2840696117100832.

log function

Syntax	<code><double> math: log (< int long float double > number, < int long float double > base)</code>
Extension Type	Function
Description	Returns the logarithm (base=base) of <code>number</code> .
Example	<code>log(34, 2f)</code> returns 5.08746284125034.

Max function

Syntax	<code><double> math: max (< int long float double > p1, < int long float double > p2)</code>
Extension Type	Function
Description	Returns the greater value out of <code>p1</code> and <code>p2</code> .
Example	<code>max(123.67d, 91)</code> returns 123.67.

Min function

Syntax	<code><double> math: min (< int long float double > p1, < int long float double > p2)</code>
Extension Type	Function
Description	Returns the smaller value out of <code>p1</code> and <code>p2</code> .
Example	<code>min(123.67d, 91)</code> returns 91.

Octal function

Syntax	<code><string> math: oct (<int long> p1)</code>
Extension Type	Function
Description	Converts <code>p1</code> to octal.
Example	<code>oct(991)</code> returns "143".

Parse Double function

Syntax	<code><double> math: parseDouble (<string> str)</code>
Extension Type	Function
Description	Returns <code>str</code> as a double.
Example	<code>parseDouble("123")</code> returns 123.0.

Parse Float function

Syntax	<code><float> math: parseFloat (<string> str)</code>
Extension Type	Function
Description	Returns <code>str</code> as a float.
Example	<code>parseFloat("123")</code> returns 123.0.

Parse Int function

Syntax	<code><int> math: parseInt (<string> str)</code>
Extension Type	Function
Description	Returns <code>str</code> as an int.
Example	<code>parseInt("123")</code> returns 123.

Parse Long function

Syntax	<code><long> math: parseLong (<string> str)</code>
Extension Type	Function
Description	Returns <code>str</code> as a long.
Example	<code>parseLong("123")</code> returns 123.

Percentile function

Syntax	<code><double> math: percentile (<int long float double> arg , <double> p)</code>
---------------	------------------------------------------------------------------------------------------------------

Extension Type	Function
Description	Returns the pth percentile value of the arg values.
Example	<pre>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.</pre>

pi function

Syntax	<code><double> math: pi ()</code>
Extension Type	Function
Description	Returns the <code>java.lang.Math.PI</code> constant, which is the closest value to pi (i.e. the ratio of the circumference of a circle to its diameter).
Example	<code>pi()</code> always returns 3.141592653589793.

Power function

Syntax	<code><double> math: power (< int long float double> value, <int long float double> toPower)</code>
Extension Type	Function
Description	Returns <code>value</code> raised to the power of <code>toPower</code> .
Example	<code>power(5.6d, 3.0d)</code> returns 175.61599999999996.

Random function

Syntax	<code><double> math: rand ()</code>
Extension Type	Function
Description	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.
Example	Two sequential calls to <code>rand()</code> may return 0.8263929447650588 and 0.24425883860361197 respectively.
Syntax	<code><double> math: rand (< int long > seed)</code>
Extension Type	Function
Description	A sequence of calls to <code>rand(seed)</code> generates a stream of pseudo-random numbers. This function uses the <code>java.util.Random</code> class internally.

Example	Two sequential calls to <code>rand(12)</code> may return <code>0.7298928061101974</code> and <code>0.2750691655200749</code> , respectively.
----------------	----------------------------------------------------------------------------------------------------------------------------------------------

Round function

Syntax	<code><int> math: round (<float> value)</code>
Extension Type	Funcion
Description	Returns the closest integer value to the argument.
Example	<code>round(3.35)</code> returns <code>3</code> .

Syntax	<code><long> math: round (<double> value)</code>
Extension Type	Function
Description	Returns the closest long value to the argument.
Example	<code>round(3252.353)</code> returns <code>3252</code> .

Sign of Number function

Syntax	<code><int> math: signum (< int long float double > p1)</code>
Extension Type	Function
Description	<ul style="list-style-type: none"> If <code>a</code> is a positive, this returns the sign of <code>p1</code> as <code>1.0</code>. If <code>a</code> is a negative, this returns the sign of <code>p1</code> as <code>-1.0</code>. If <code>a</code> is neither a positive or a negative, this returns the sign of <code>p1</code> as <code>0.0</code>. <p>This function wraps the <code>java.lang.Math.signum()</code> function.</p>
Example	<code>signum(-6.32d)</code> returns <code>-1</code> .

sin function

Syntax	<code><double> math: sin (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the sine of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.sin()</code> function.
Example	<code>sin(6d)</code> returns <code>-0.27941549819892586</code> .

sinh function

Syntax	<code><double> math: sinh (< int long float double > p1)</code>
Extension Type	Function

Description	Returns the hyperbolic sine of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.sinh()</code> function.
Example	<code>sinh(6d)</code> returns <code>201.71315737027922</code> .

Square Root function

Syntax	<code><double> math: sqrt (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the square-root of <code>p1</code> . This function wraps the <code>java.lang.Math.sqrt()</code> function.
Example	<code>sqrt(4d)</code> returns <code>2</code> .

`tan` function

Syntax	<code><double> math: tan (< int long float double > p1)</code>
Extension Type	Function
Description	Returns the <code>tan</code> of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.tan()</code> function.
Example	<code>tan(6d)</code> returns <code>-0.29100619138474915</code> .

`tanh` function

Syntax	<code><double> math: tanh (<int long float double> p1)</code>
Extension Type	Function
Description	Returns the hyperbolic tangent of <code>p1</code> (<code>p1</code> is in radians). This function wraps the <code>java.lang.Math.tanh()</code> function.
Example	<code>tanh(6d)</code> returns <code>0.9999877116507956</code> .

To Degrees function

Syntax	<code><double> math: toDegrees (< int long float double > p1)</code>
Extension Type	Function
Description	Converts <code>p1</code> from radians to degrees. This function wraps the <code>java.lang.Math.toDegrees()</code> function.
Example	<code>toDegrees(6d)</code> returns <code>343.77467707849394</code> .

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 <code>java.lang.Math.toRadians()</code> function.
Example	<code>toRadians(6d)</code> 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
- Upper Case function

Char At function

Syntax	<string> str:charAt(<string> str, <int> index)
Extension Type	Function
Description	Returns the char value as a string value at the specified index.
Example	<code>charAt("WSO2", 1)</code> returns 'S'.

Coalesce function

Syntax	< int long float double string boolean > str: coalesce (< 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 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	<ul style="list-style-type: none"> • <code>coalesce("123", null, "789")</code> returns "123". • <code>coalesce(null, "BBB", "CCC")</code> returns "BBB". • <code>coalesce(null, null, null)</code> returns null.

Concatenation function

Syntax	<code><string> str: concat (<int long float double string boolean > arg1, < int long float double string boolean > arg2 ,..., < int long float double string boolean > argN)</code>
Extension Type	Function
Description	Returns a string that is the result of concatenating the given arguments: <code>arg1, arg2,.., argN</code> .
Examples	<ul style="list-style-type: none"> • <code>concat("D533", "8JU^", "XYZ")</code> returns "D5338JU^XYZ". • <code>concat("AAA", null, "CCC")</code> returns "AAACCC".

Contains function

Syntax	<code><bool> str: contains (< string> inputSequence, <string> searchingSequence)</code>
Extension Type	Function
Description	This method returns true if <code>inputSequence</code> contains the specified sequence of char values in the <code>searchingSequence</code> .
Example	<code>contains("21 products are produced by WSO2 currently", "WSO2")</code> returns true.

Hexadecimal function

Syntax	<code><string> str: hex (< string> str)</code>
Extension Type	Function
Description	Returns a hexadecimal string representation of <code>str</code> .
Example	<code>hex("MySQL")</code> returns "4d7953514c".

Length function

Syntax	<code><int> str: length (< string> str)</code>
Extension Type	Function
Description	Returns the length of the string: <code>str</code> .

Examples	length("Hello World") returns 11.
-----------------	-------------------------------------

Lower Case function

Syntax	<code><string> str: lower (< string> str)</code>
Extension Type	Function
Description	Converts the capital letters in the <code>str</code> input string to the equivalent simple letters.
Example	<code>lower("WSO2 cep") returns "wso2 cep".</code>

Regular Expression function

Syntax	<code><boolean> str: regexp (< string> str, <string> regex)</code>
Extension Type	Function
Description	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.
Example	<code>regexp("WSO2 abcdh" , "WSO(.*)h") returns true.</code>

Repeat function

Syntax	<code><string> str: repeat (< string> str, <int> times)</code>
Extension Type	Function
Description	Repeats the specified string (i.e. <code>string</code>) for the specified number of times (i.e. <code>times</code>).
Example	<code>repeat("StRing 1" , 3) returns "StRing 1StRing 1StRing 1".</code>

Replace All function

Syntax	<code><string> str: replaceAll (< string> str, <string> regex , <string> replacement)</code>
Extension Type	Function
Description	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>).
Example	<code>replaceAll("hello hi hello" , 'hello' , 'test') returns "test hi test".</code>

Replace First function

Syntax	<code><string> str: replaceFirst (< string> str , <string> regex , <string> replacement)</code>
Extension Type	Function

Description	Replaces the first substring that matches the given regular expression (i.e. <code>regex</code>) with the string specified as the replacement (i.e. <code>replacement</code>)
Example	<code>replaceFirst("hello WSO2 A hello", 'WSO2(.*)A', 'XXXX') returns "hello XXXX hello".</code>

Reverse function

Syntax	<code><string> str: reverse (< string> str)</code>
Extension Type	Function
Description	Returns the reverse ordered string of <code>str</code> .
Example	<code>reverse("Hello World") returns "dlrow olleH".</code>

Split function

Syntax	<code><string> str: split (<string> sourceString, <string> splitCharacter, <int> returnedOutputPosition)</code>
Extension Type	Function
Description	This method splits the given <code>sourceString</code> by given <code>splitCharacter</code> and returns the string in the index given by <code>returnedOutputPosition</code> .
Example	<code>split("WSO2,ABM,NSFT", "", 0) returns WSO2.</code>

String Compare function

Syntax	<code><int> str: strcmp (< string> str, <string> compareTo)</code>
Extension Type	Function
Description	Compares <code>str</code> with <code>compareTo</code> strings lexicographically.
Examples	<ul style="list-style-type: none"> <code>strcmp("Hello", "Hello")</code> returns 0. <code>strcmp("AbCDefghiJ KLMN", "Hello")</code> returns -7.

Sub String function

Syntax	<code><string> str: substr (< string> sourceText, <int> beginIndex)</code>
Extension Type	Function
Description	Returns a new string that is a substring of <code>sourceText</code> .
Example	<code>substr("AbCDefghiJ KLMN", 4) returns "efghiJ KLMN".</code>
Syntax	<code><string> str: substr (< string> sourceText, <int> beginIndex, <int> length)</code>

Extension Type	Function
Description	Returns a new string that is a substring of <code>sourceText</code> .
Example	<code>substr("AbCDefghiJ KLMN", 2, 4) returns "CDef".</code>

Syntax	<code><string> str: substr (< string> sourceText, <string> regex)</code>
Extension Type	Function
Description	Returns a new string that is a substring of <code>sourceText</code> .
Examples	<code>substr("WSO2D efghiJ KLMN", '^WSO2(.*)') returns "WSO2D efghiJ KLMN".</code>

Syntax	<code><string> str: substr (< string> sourceText, <string> regex, <int> groupNumber)</code>
Extension Type	Function
Description	Returns a new string that is a substring of <code>sourceText</code> .
Example	<code>substr("WSO2 cep WSO2 XX E hi hA WSO2 heHello", 'WSO2(.*)A(.*)', 2) returns " ello".</code>

Trim function

Syntax	<code><string> str: trim (< string> str)</code>
Extension Type	Function
Description	Returns a copy of <code>str</code> , with the leading and/or trailing white-spaces omitted.
Example	<code>trim(" AbCDefghiJ KLMN ") returns "AbCDefghiJ KLMN".</code>

Unhexadecimal function

Syntax	<code><string> str: unhex (< string> str)</code>
Extension Type	Function
Description	This is the equivalent of the <code>unhex</code> function in mysql 5.0. <code>unhex(str)</code> interprets each pair of characters in <code>str</code> as a hexadecimal number. Also see hex () string extension.
Example	<code>unhex("4d7953514c") returns "MySQL".</code>

Upper Case function

Syntax	<code><string> str: upper (<string> str)</code>
Extension Type	Function

Description	Converts the simple letters in the given input string (i.e. <code>str</code>) to the equivalent capital letters.
Example	<code>upper("Hello World") returns "HELLO WORLD".</code>

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

Intersects function

Syntax	<code><bool> geo:intersects (<string> geoJSONGeometry , <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns <code>true</code> if the <code>geoJSONGeometry</code> incoming event intersects the given string (i.e., <code>geoJSONGeometryFence</code>). Returns <code>false</code> otherwise.
Example	<code>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]]]})</code> , returns <code>true</code> because <code>geoJSONGeometry</code> intersects <code>geoJSONGeometryFence</code> .

Syntax	<code><bool> geo: intersects (<double> longitude , <double> latitude , <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns <code>true</code> if the location specified in terms of longitude and latitude intersects the given <code>geoJSONGeometryFence</code> . Returns <code>false</code> otherwise.
Example	<code>intersects(0.5, 0.5 , {'type':'Polygon','coordinates':[[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]]})</code> , returns <code>true</code> because the location specified in terms of longitude and latitude intersects <code>geoJSONGeometryFence</code> .

Within function

Syntax	<code><bool> geo: within (<double> longitude , <double> latitude, <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns <code>true</code> if the location specified in terms of longitude and latitude is within the <code>geoJSONGeometryFence</code> .

Examples	<ul style="list-style-type: none"> 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	<code><bool> geo: within (<string> geoJSONGeometry , <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns true if the <code>geoJSONGeometry</code> is within the <code>geoJSONGeometryFence</code> . Returns false otherwise.
Example	<ul style="list-style-type: none"> 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

Syntax	<code><bool> geo: withindistance (<double> longitude , <double> latitude, <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	Returns <code>true</code> if the location specified in terms of longitude and latitude is within distance of the <code>geoJSONGeometryFence</code> . Returns <code>false</code> otherwise.
Example	<code>withindistance(0.5 , 0.5, {'type':'Polygon','coordinates':[[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]]}, 110574.61087757687)</code> returns <code>true</code> because the location specified in terms of longitude and latitude is within the distance of the <code>geoJSONGeometryFence</code> .

Syntax	<code><bool> geo: withindistance (<string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> distance)</code>
Extension Type	Function
Description	Returns <code>true</code> if the area given by <code>geoJSONGeometry</code> is within distance of the <code>geoJSONGeometryFence</code> .
Example	<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]]]}, 110574.61087757687)</code> returns <code>true</code> because <code>geoJSONGeometry</code> is within the distance of <code>geoJSONGeometryFence</code> .

Crosses function

Syntax	<code><bool> geo: crosses (<string> id , <double> longitude, <double> latitude , <string> geoJSONGeometryFence)</code>
Extension Type	Function
Description	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> .
Example	<ul style="list-style-type: none"> <code>crosses(km-4354, -0.5, 0.5, {'type':'Polygon', 'coordinates':[[[0, 0],[2, 0],[2, 1],[0, 1],[0, 0]]}])</code> returns <code>true</code>. <code>km-4354, 1.5, 0.5, {'type':'Polygon', 'coordinates':[[[0, 0],[2, 0],[2, 1],[0, 1],[0, 0]]]}</code> returns <code>true</code>.

Syntax	<code><bool> geo: crosses (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence)</code>
Extension Type	StreamProcessor
Description	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> .

Stationary function

Syntax	<code><bool> geo: stationary (<string> id , <double> longitude, <double> latitude , <string> geoJSONGeometryFence , <double> radius)</code>
Extension Type	StreamProcessor
Description	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> .
Example	<ul style="list-style-type: none"> <code>stationary(km-4354,0,0, 110574.61087757687)</code> returns <code>true</code>. <code>stationary(km-4354,1,1, 110574.61087757687)</code> returns <code>true</code>. <code>stationary(km-4354,1,1.5, 110574.61087757687)</code> returns <code>true</code>.

Syntax	<code><bool> geo: stationary (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> radius)</code>
Extension Type	StreamProcessor
Description	Returns <code>true</code> when the object (i.e. <code>geoJSONGeometry</code>) becomes stationary within the specified <code>radius</code> . Returns <code>false</code> when the objects moves out of the specified radius.

Proximity function

Syntax	<bool, string> geo:proximity (<string> id , <double> longitude , <double> latitude , <string> geoJSONGeometryFence , <double> radius)
Extension Type	StreamProcessor
Description	Returns <code>true</code> when two objects (specified in terms of <code>longitude</code> and <code>latitude</code>) are within the specified <code>radius</code> to another object. Returns <code>false</code> when the specified object moves out of the specified <code>radius</code> . The <code>proximityWith</code> optional attribute indicates the ID of the object that the object specified is in close proximity with. <code>proximityID</code> is a unique ID for the two objects in close proximity.
Example	The following return <code>true</code> with <code>id 3</code> . <ul style="list-style-type: none"> • <code>proximity(1, 0, 0, 110574.61087757687)</code> • <code>proximity(2, 1, 1, 110574.61087757687)</code> • <code>proximity(3, 2, 2, 110574.61087757687)</code> • <code>proximity(1, 1.5, 1.5, 110574.61087757687)</code>

Syntax	<bool> proximity (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> radius)
Extension Type	StreamProcessor
Description	Returns <code>true</code> when an object (i.e. <code>geoJSONGeometry</code>) is within the specified <code>radius</code> from another object. Returns <code>false</code> when one or both objects move away from each other and are no longer within the specified <code>radius</code> s of each other. The <code>proximityWith</code> optional attribute indicates the ID of the object that the object specified is in close proximity with. <code>proximityID</code> is a unique ID for the two objects in close proximity.

Geo Coordinates function

Syntax	<double, double, string> geocode (<string> location)
Extension Type	StreamProcessor
Description	Transforms a location to its geo-coordinates (<code>longitude</code> and <code>latitude</code>) and formatted address.
Example	<code>geocode(duplication rd)</code> returns the following data with adhering latitude, longitude, and formattedAddress attribute names respectively. 6.8995244d, 79.8556202d, "R A De Mel Mawatha, Colombo, Sri Lanka"

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> output1 , <int long float double string boolean> output2 , ...] r:eval (<string > script , <string > outputAttributes , <int long float double string boolean> input1 , <int long float double string boolean> 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. <pre>@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;</pre>

Evaluation Source function

Syntax	[<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 boolean> 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	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.
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```
@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

Syntax	<bool> regex:find (<string> regex , <string> inputSequence)
Extension Type	Function
Description	This method attempts to find the next sub-sequence of the inputSequence that matches the regex pattern. It returns true if such a sub sequence exists, or returns false otherwise.
Examples	<ul style="list-style-type: none"> • <code>find("\d\d(.*)WSO2", "21 products are produced by WSO2 currently")</code> returns true. • <code>find("\d\d(.*)WSO2", "21 products are produced currently")</code> returns false.

Syntax	<bool> regex: find (<string> regex , <string> inputSequence , <int> startIndex)
Extension Type	Function
Description	This method attempts to find the next sub-sequence of the inputSequence that matches the regex pattern starting from given index (i.e. startIndex). It returns true if such a sub sequence exists, or returns false otherwise.
Examples	<ul style="list-style-type: none"> • <code>find("\d\d(.*)WSO2", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 30)</code> returns true. • <code>find("\d\d(.*)WSO2", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 35)</code> returns false.

Group function

Syntax	<code><string> regex: group (<string> regex , <string> inputSequence , <int> groupId)</code>
Extension Type	Function
Description	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 .
Example	<code>group("(\\d\\d)(.*)(WSO2.*)", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 3)</code> returns "WSO2 employees".

Looking At function

Syntax	<code><string> regex: lookingAt (<string> regex , <string> inputSequence)</code>
Extension Type	Function
Description	This method attempts to match the <code>inputSequence</code> against the <code>regex</code> pattern starting at the beginning.
Examples	<ul style="list-style-type: none"> <code>lookingAt("\d\d(.*)WSO2", "21 products are produced by WSO2 currently in Sri Lanka")</code> returns true. <code>lookingAt("WSO2(.*)middleware(.*)", "sample test string and WSO2 is situated in trace and its a middleware company")</code> returns false.

Matches function

Syntax	<code><string> regex: matches (<string> regex , <string> inputSequence)</code>
Extension Type	Function
Description	This method attempts to match the entire <code>inputSequence</code> against the <code>regex</code> pattern.
Examples	<ul style="list-style-type: none"> <code>matches("WSO2(.*)middleware(.*)", "WSO2 is situated in trace and its a middleware company")</code> returns true. <code>matches("WSO2(.*)middleware", "WSO2 is situated in trace and its a middleware company")</code> 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
- Timestamp In Milliseconds function

- UTC Timestamp function

Current Date function

Syntax	<code><string> time:currentDate ()</code>
Extension Type	Function
Description	Returns the current system date in the <code>yyyy-MM-dd</code> format.
Example	<code>currentDate()</code> returns <code>2015-08-20</code> .

Current Time function

Syntax	<code><string> currentTime ()</code>
Extension Type	Function
Description	Returns the current system time in the <code>HH:mm:ss</code> format.
Example	<code>currentTime()</code> returns <code>13:15:10</code> .

Current Timestamp function

Syntax	<code><string> time: currentTimestamp ()</code>
Extension Type	Function
Description	Returns the current system timestamp in the <code>yyyy-MM-dd HH:mm:ss</code> format.
Example	<code>currentTimestamp()</code> returns <code>2015-08-20 13:15:10</code> .

Date Adding function

The common parameters of this function are described below.

- **dateValue** : A value of date. e.g., `"2014-11-11 13:23:44.657"`, `"2014-11-11"`, `"13:23:44.657"`
- **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	<code><string> time: dateAdd (<string> dateValue , <long> expr, <string> unit, <string> dateFormat)</code>
Extension Type	Function
Description	Returns the specified date and time with the selected <code>unit</code> of the specified <code>dateValue</code> incremented by the given amount (i.e. <code>expr</code>).
Example	<code>dateAdd("2014-11-11 13:23:44", 2, 'year', "yyyy-MM-dd HH:mm:ss")</code> returns <code>"2016-11-11 13:23:44"</code> .

Syntax	<code><string> time: dateAdd (<string> dateValue , < long > expr, <string> unit)</code>
Extension Type	Function
Description	Returns the specified date and time with the selected <code>unit</code> of the specified <code>dateValue</code> incremented by the given amount (i.e. <code>expr</code>).
Example	<code>dateAdd("2014-11-11 13:23:44", 2, 'year')</code> returns "2016-11-11 13:23:44".

Syntax	<code><string> time: dateAdd (<long> timestampInMilliseconds, < long > expr, <string> unit)</code>
Extension Type	Function
Description	Returns the specified time stamp with the selected <code>unit</code> of the specified <code>timestampInMilliseconds</code> incremented by the given amount (i.e. <code>expr</code>).
Example	<code>dateAdd(1415692424000L, 2, 'year')</code> returns "2016-11-11 13:23:44".

Date Subtraction function

The common parameters of this function are described below.

- `dateValue` : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- `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	<code><string> time: dateSub (<string> dateValue , <long> expr, <string> unit, <string> dateFormat)</code>
Extension Type	Function
Description	Returns the specified date and time with the selected <code>unit</code> of the specified <code>dateValue</code> reduced by the given amount (i.e. <code>expr</code>).
Example	<code>dateSub("2014-11-11 13:23:44", 2, 'year', "yyyy-MM-dd HH:mm:ss")</code> returns "2012-11-11 13:23:44".

Syntax	<code><string> time: dateSub (<string> dateValue , < long > expr, <string> unit)</code>
Extension Type	Function
Description	Returns the specified date and time stamp with the selected <code>unit</code> of the specified <code>dateValue</code> reduced by the given amount (i.e. <code>expr</code>).
Example	<code>dateSub("2014-11-11 13:23:44", 2, 'year')</code> returns "2012-11-11 13:23:44".

Syntax	<code><string> time: dateSub (<long> timestampInMilliseconds, < long > expr , <string> unit)</code>
Extension Type	Function
Description	Returns the specified time stamp with the selected <code>unit</code> of the specified <code>timestampInMilliseconds</code> reduced by the given amount (i.e. <code>expr</code>).
Example	<code>dateSub(1415692424000L, 2, 'year')</code> returns 1352620424000.

Date Difference function

The common parameters of this function are described below.

- `dateValue1` : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11" , "13:23:44.657"
- `dateValue2` : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11" , "13:23:44.657"
- `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	<code><int> time: dateDiff (<string> dateValue1 , < string > dateValue2 , <string> dateFormat1, <string> dateFormat2)</code>
Extension Type	Function
Description	Returns the number of days between the two dates specified (i.e. <code>dateValue1</code> and <code>dateValue2</code>).
Example	<code>dateDiff('2014-11-11 13:23:44', '2014-11-9 13:23:44', 'yyyy-MM-dd HH:mm:ss', 'yyyy-MM-dd HH:mm:ss')</code> returns 2.

Syntax	<code><int> time: dateDiff (<string> dateValue1 , < string > dateValue2)</code>
Extension Type	Function
Description	Returns the number of days between the two dates specified (i.e. <code>dateValue1</code> and <code>dateValue2</code>).
Example	<code>dateDiff('2014-11-11 13:23:44.000', '2014-11-9 13:23:44.000')</code> returns 2.

Syntax	<code><int> time: dateDiff (<string> timestampInMilliseconds1 , < string > timestampInMilliseconds2)</code>
Extension Type	Function
Description	Returns the number of days between the two date and time stamps specified (i.e. <code>timestampInMilliseconds1</code> and <code>timestampInMilliseconds2</code>).
Example	<code>dateDiff(1415692424000, 1415519624000)</code> returns 2.

Date Format function

The common parameters of this function are described below.

- **dateValue** : -A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- **dateTargetFormat** : The date format to which the specified **date value** needs to be converted. e.g., yy
yy/MM/dd HH:mm:ss
- **dataSourceFormat** : 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	<string> time: dateFormat (<string> dateValue ,<string> dateTargetFormat , <string> dataSourceFormat)
Extension Type	Function
Description	Returns a formatted date string.
Example	dateFormat('2014-11-11 13:23:55', 'ss', 'yyyy-MM-dd HH:mm:ss') returns 55.

Syntax	<string> time: dateFormat (<string> dateValue ,<string> dateTargetFormat)
Extension Type	Function
Description	Returns a formatted date string.
Example	dateFormat('2014-11-11 13:23:55.657', 'ss') returns 55.

Syntax	<string> time: dateFormat (<long> timestampInMilliseconds ,<string> dateTargetFormat)
Extension Type	Function
Description	Returns a formatted date string.
Example	dateFormat(1415692424000, 'yyyy-MM-dd') returns 2014-11-11.

Extract function

- **dateValue** : A value of date. e.g., "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- **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** : A date value in milliseconds (from the epoch) e.g., 1415712224000L

Syntax	<int> time: extract (<string> unit ,<string> dateValue , <string> dateFormat)
Extension Type	Function
Description	Returns the specified unit extracted from the specified dateValue .
Example	extract('year', '2014-3-11 02:23:44', 'yyyy-MM-dd hh:mm:ss') returns 2014.

Syntax	<code><int> time: extract (<string> unit ,<string> dateValue)</code>
Extension Type	Function
Description	Returns the specified <code>unit</code> extracted from the specified <code>dateValue</code> .
Example	<code>extract('year', '2014-3-11 02:23:44.234') returns 2014.</code>

Syntax	<code><int> time: extract (<long> timestampInMilliseconds ,<string> unit)</code>
Extension Type	Function
Description	Returns the specified <code>unit</code> extracted from the specified <code>timestampInMilliseconds</code> .
Example	<code>extract(1394484824000, 'year') returns 2014.</code>

Date function

Syntax	<code><string> time: date (<string> dateValue ,<string> dateFormat)</code>
Extension Type	Function
Description	Returns the date component of the <code>dateValue</code> .
Example	<code>date('2014-11-11 13:23:44', 'yyyy-MM-dd HH:mm:ss') returns 2014-11-11.</code>

Timestamp In Milliseconds function

Syntax	<code><long> time: timestampInMilliseconds ()</code>
Extension Type	Function
Description	Returns the current time stamp in milliseconds.
Example	<code>timestampInMilliseconds() returns 1440160328693.</code>

Syntax	<code><long> time: timestampInMilliseconds (<string> dateValue)</code>
Extension Type	Function
Description	Returns the time stamp of the specified <code>dateValue</code> in milliseconds. In order to use this function, the date format of the specified <code>dateValue</code> should be <code>yyyy-MM-dd HH:mm:ss.SSS</code> .
Example	<code>timestampInMilliseconds('2007-11-30 10:30:19.000') returns 1196398819000.</code>

Syntax	<code><long> time: timestampInMilliseconds (<string> dateValue, <string> dateFormat)</code>
Extension Type	Function
Description	Returns the time stamp of the specified <code>dateValue</code> in milliseconds. The date format can be specified in the <code>dateFormat</code> parameter.

Example	<code>timestampInMilliseconds('2007-11-30 10:30:19', 'yyyy-MM-dd HH:mm:ss') returns 1196398819000.</code>
----------------	-----------------------------------------------------------------------------------------------------------

UTC Timestamp function

Syntax	<code><string> time: utcTimestamp()</code>
Extension Type	Function
Description	Returns the system time in the <code>yyyy-MM-dd HH:mm:ss</code> date format.
Example	<code>utcTimestamp()</code> returns <code>2015-08-21 12:16:13</code> .

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
- Find Name Entity Type Via Dictionary function
- Find Relationship By Verb function
- Find Relationship By Regex function
- Find Semgrex Pattern function
- Find Tokens Regex Pattern function

Find Name Entity Type function

Syntax	<code><string> nlp:findNameEntityType(<string> entityType, <bool> groupSuccessiveMatch, <string> string-variable)</code>
Extension Type	Function
Description	<p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • <code>entityType</code> : This is a user-specified string constant. e.g., PERSON, LOCATION, ORGANIZATION, MONEY, PERCENT, DATE or TIME • <code>groupSuccessiveMatch</code> : This is a user-specified boolean constant used to group successive matches of the specified <code>entityType</code> and a text stream. • <code>streamAttribute</code> : A string or the stream attribute in which text stream is included. <p>This function returns the entities in the text. If you specify group successive matches as <code>true</code>, the result aggregates successive words of the same entity type.</p>
Example	<pre>findNameEntityType("PERSON", true, text)</pre> <p>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.</p>

Find Name Entity Type Via Dictionary function

Syntax	<code><string> nlp: findNameEntityTypeViaDictionary(<string> entityType, <string> dictionaryFilePath, <string> string-variable)</code>
Extension Type	Function

Description	<p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> 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. <pre><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>WS02</entry> </entity> </dictionary></pre> <ul style="list-style-type: none"> streamAttribute : A string or the stream attribute in which text stream is included. <p>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.</p>
Example	<pre>findNameEntityTypeViaDictionary("PERSON" , "dictionary.xml" ,text)</pre> <p>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".</p>

Find Relationship By Verb function

Syntax	<pre><string > text, <string> subject, < string > object, < string > verb nlp: findRelationshipByVerb (<string> verb, <string> string-variable)</pre>
Extension Type	Function
Description	<p><code>findRelationshipByVerb</code> 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.</p> <p>The input parameters used are as follows.</p> <ul style="list-style-type: none"> verb : This is a user specified string constant. string-variable : A string or the stream attribute which includes the text stream.
Examples	<pre>findRelationshipByVerb("say", "Information just reaching us says another Liberian With Ebola Arrested At Lagos Airport") returns the following.</pre> <ul style="list-style-type: none"> The whole text Information as the subject Liberian as the object. says as the verb.

Find Relationship By Regex function

Syntax	<code><string > text, <string> subject, < string > object, < string > verb nlp:findRelationshipByRegex (<string> regex, <string> string-variable)</code>
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 Semgrep pattern.
Example	<pre>findRelationshipByRegex('{}=verb >/nsubj agent/ {}=subject >/dobj/ {}=object', "gates foundation donates \$50M in support of #Ebola relief")</pre> <p>returns the following.</p> <ul style="list-style-type: none"> • The whole text • "foundation" as the subject • "\$" as the object • "donates" as the verb

Find Semgrep Pattern function

Syntax	<code><string > text, <string> match, < string > object, < string > verb nlp: findSemgrepPattern (<string> regex, <string> string-variable)</code>
Extension Type	Function
Description	<p>The <code>findSemgrepPattern</code> function returns the whole text, subject, object and verb from the text stream that matches the named nodes of the Semgrep pattern.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • regex : A user specified regular expression that matches the Semgrep pattern syntax. • string-variable : A string or the stream attribute which includes the text stream.
Example	<pre>findSemgrepPattern('{}lemma:die} >.*subj num.*/=reln {}=diedsubject', "Sierra Leone doctor dies of Ebola after failed evacuation.")</pre> <p>In this example, the function searches for words with the lemmatization <code>die</code> that are governors on any subject or numeric relation. The dependent is marked as the <code>diedsubject</code>, and the relationship is marked as <code>reln</code>. Thus, the query returns an output stream that has the full match of this expression, i.e. the governing word with lemmatization for <code>die</code>. It also returns the name of the corresponding node for each match it finds.</p> <p>The following is the list of elements in the output stream.</p> <ul style="list-style-type: none"> • The whole text • dies as the match • "nsubj" as reln • doctor asdiedsubject

Find Tokens Regex Pattern function

Syntax	<code>< string > text, <string> match, <string> group_1, etc. nlp: findTokensRegexPattern (<string> regex, <string> string-variable)</code>
Extension Type	Function

Description	<p><code>findTokensRegexPattern</code> 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.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • 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	<pre>findTokensRegexPattern('([ner:/PERSON ORGANIZATION LOCATION/]*) ([?:[]* [1 emma:donate]) ([ner: MONEY]+)', text) defines three groups:</pre> <ul style="list-style-type: none"> • 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. <p>This function returns the following.</p> <ul style="list-style-type: none"> • 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	<pre><Object> map:create()</pre> <p>or</p> <pre><Object> map:create(<Object> key1, <Object> value1, <Object> key2, <Object> value2, ... <Object> keyN, <Object> valueN)</pre>
Extension Type	Function
Description	Returns the created map object.
Examples	<ul style="list-style-type: none"> • <code>create()</code> returns an empty map. • <code>create(1, "one", 2, "two", 3, "three")</code> returns a map with keys 1, 2, 3 and corresponding values "one", "two", "three".

Get function

Syntax	<Object> map:get (<Map> map, <Object> key)
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

Syntax	<bool> map:isMap (<Object> object)
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

Syntax	<Object> map:put (<Object> map, <Object> key, <Object> value)
Extension Type	Function
Description	Returns the updated map after adding the given key-value pair.
Example	put(students, 1234, "sam") returns the updated map named students after adding the object "sam" with key 1234.

Remove function

Syntax	<Object> map:remove (<Object> map, <Object> key)
Extension Type	Function
Description	Returns the updated map after removing the element with key.
Example	remove(students, 1234) returns the updated map students after removing the element with the key 1234.

Create from JSON function

Syntax	<Object> map:createFromJSON (<string> JSONstring)
Extension Type	Function
Description	Returns the map created with the key values pairs given in the JSONstring.

Example	<code>createFromJSON("{'symbol' : 'IBM' , 'price' : 200, 'volume' : 100}")</code> returns a map with the keys "symbol", "price", "volume", and with the values "IBM", 200 and 100 respectively.
----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Create from XML function

Syntax	<code><Object> map:createFromXML(<string> XMLstring)</code>
Extension Type	Function
Description	Returns the map created with the key values pairs given in the XMLstring.
Example	<code>createFromXML("<company> <symbol> wso2 </symbol> <price> <100> </price> <volume> 200 </volume> </company>")</code> returns a map with the keys "symbol", "price", "volume", and with the values WSO2, 100 and 200 respectively.

To JSON function

Syntax	<code><String> map:toJSON(<Object> map)</code>
Extension Type	Function
Description	Converts a map into a JSON object and returns the definition of that JSON object as a string.
Example	If "company" is a map with key value pairs ("symbol" : wso2), ("volume" : 100), and ("price", 200), <code>toJSON(company)</code> returns the string <code>{"symbol" : "wso2" , "volume" : 100 , "price" : 200}</code> .

To XML function

Syntax	<code><String> map:toXML(<Object> map)</code> or <code><String> map:toXML(<Object> map, <String> rootElementName)</code>
Extension Type	Function
Description	Returns the map as an XML string.
Example	If "company" is a map with key value pairs ("symbol" : wso2), ("volume" : 100), and ("price", 200), <code>toXML(company)</code> returns the string <code><symbol>wso2</symbol><volume><100></volume><price>200</price></code> . <code>toXML(company, "abcCompany")</code> returns the string <code><abcCompany><symbol>wso2</symbol><volume><100></volume><price>200</price></abcCompany></code> .

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.

```
@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**

K-Slack function

Syntax	<code><bool> reorder:kslack(<long> timestamp)</code>
Extension Type	StreamProcessor
Description	This is the most basic version. The events are sorted by the <code>timestamp</code> parameter.

Syntax	<code><bool> reorder:kslack(<long> timestamp, <long> timeOut)</code>
Extension Type	StreamProcessor
Description	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.

Syntax	<code><bool> reorder:kslack(<long> timestamp, <long> timeOut, <long> maxValue)</code>
Extension Type	StreamProcessor
Description	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.
Example	<ul style="list-style-type: none"> • <code>kslack(timestamp, -11, 1000000)</code> 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 <code>-11</code> which effectively disables the timer based draining of the internal buffer. • <code>kslack(timestamp, 10001, 1000000)</code> 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	<code><bool> reorder:kslack (<long> timestamp, <long> timeOut, <bool> expireFlag)</code>
Extension Type	StreamProcessor
Description	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.
Example	<code>kslack(timestamp, -11, true)</code>

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.

Parameter	Required/Optional	Description
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.
markovMatrixStorageLocation	Required	The location of the CSV file that contains the existing Markov matrix to be used.

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.

Parameter	Name	Description
lastState	Last state	The previous state of the particular ID.
transitionProbability	Transition probability	The transition probability between the previous state and the current state for a particular ID.
notify	notify	This signifies a notification that indicates that the transition probability is less than or equal to the alert threshold probability.

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.

```
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.

```
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.

Parameter	Required/Optional	Description
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.
notificationsHoldLimiter	Required	The number of events that should be received before the matrix starts triggering notifications.
train	Optional	If this is set to <code>true</code> , event values are used to train the Markov matrix. If this is set to <code>false</code> , the Markov matrix values remain the same.

Output parameters

The following are the output parameters for this extension.

Parameter	Name	Description
lastState	Last state	The previous state of the particular ID.
transitionProbability	Transition probability	The transition probability between the previous state and the current state for a particular ID.
notify	notify	This signifies a notification that indicates that the transition probability is less than or equal to the alert threshold probability.

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.

```
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 > <code>pmmml:predict(<string> pathToPmmlFile)</code>
Extension Type	Stream Processor

Description	<p>Processes the input stream attributes according to the defined PMML standard model and outputs the processed results together with the input stream attributes.</p> <p>This function uses the following input parameter.</p> <ul style="list-style-type: none"> • pathToPmmlFile : The path to the PMML model file. <p>The function returns the outputs defined in the output fields. The number of outputs can vary.</p>
Example	<pre>predict('<CEP HOME>/samples/artifacts/0301/decision-tree.pmml')</pre> <p>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.</p>

Syntax	<code>< double float long int string boolean > pmml:predict(<string> pathToPmmlFile, <double float long int string boolean> input)</code>
Extension Type	Stream Processor
Description	<p>Processes the input stream attributes according to the defined PMML standards model and outputs the processed results.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • 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. <p>This function returns the processed outputs defined in the query. The number of outputs can vary depending on the query definition.</p>
Examples	<pre>predict('<CEP HOME>/samples/artifacts/0301/decision-tree.pmml', root_shel1, 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)</pre> <p>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.</p>

Machine Learning 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	<code>< double float long int string boolean > pmml:predict(<string> pathToPmmlFile)</code>
Extension Type	Stream Processor

Description	<p>Processes the input stream attributes according to the defined PMML standard model and outputs the processed results together with the input stream attributes.</p> <p>This function uses the following input parameter.</p> <ul style="list-style-type: none"> • pathToPmmlFile : The path to the PMML model file. <p>The function returns the outputs defined in the output fields. The number of outputs can vary.</p>
Example	<pre>predict('<CEP HOME>/samples/artifacts/0301/decision-tree.pmml')</pre> <p>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.</p>

Syntax	<code>< double float long int string boolean > pmml:predict(<string> pathToPmmlFile, <double float long int string boolean> input)</code>
Extension Type	<p>Stream Processor</p>
Description	<p>Processes the input stream attributes according to the defined PMML standards model and outputs the processed results.</p> <p>This function uses the following input parameters.</p> <ul style="list-style-type: none"> • 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. <p>This function returns the processed outputs defined in the query. The number of outputs can vary depending on the query definition.</p>
Examples	<pre>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)</pre> <p>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.</p>

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.

Syntax	<double, double> kf:kalmanFilter(<double> measuredValue)
Extension Type	Function
Example	<ul style="list-style-type: none"> • 1st round: kf:kalmanFilter(-74.178444) returns an estimated value of -74.178444. • 2nd round: kf:kalmanFilter(-74.175703) returns an estimated value of -74.1770735006853. • 3rd round: kf:kalmanFilter(-74.177872) returns an estimated value of -74.1773396670348.

Syntax	<double, double> kf: kalmanFilter(<double> measuredValue, <double> measurementNoiseSD)
Extension Type	Function
Example	<ul style="list-style-type: none"> • 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 -74.17707350205573. • 3rd round: kf:kalmanFilter(-74.177872, 0.003) returns an estimated value of -74.177339667771.

Syntax	<double, double> kf: kalmanFilter(<double> measuredValue, <double> measuredChangingRate, <double> measurementNoiseSD, <long> timestamp)
Extension Type	Function
Example	<ul style="list-style-type: none"> • 1st round: kf:kalmanFilter(-74.178444, 0.003, 0.01, time:timestampInMilliseconds()) returns an estimated value of -74.1784439700006 • 2nd round: kf:kalmanFilter(-74.178444, 0.003, 0.01, time:timestampInMilliseconds()) returns an estimated value of -74.1784439700006 • 3rd round: kf:kalmanFilter(-74.177872, 0.003, 0.01, time:timestampInMilliseconds()) returns an estimated value of -74.17697314316393.

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/DAS server. This is useful when you need to carry out complex event processing in embedded devices in which WSO2 CEP/DAS 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.

```
<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.

```
<repositories>
    <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>
</repositories>
```

You can create the Java project using any method you prefer. The required dependencies can be downloaded from [here](#).

- 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.

```
String definition = "@config(async = 'true') define stream cseEventStream (symbol
string, price float, volume long);";
```

- Define a Siddhi query as follows.

```
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.

```
SiddhiManager siddhiManager = new SiddhiManager();

ExecutionPlanRuntime executionPlanRuntime =
siddhiManager.createExecutionPlanRuntime(defination + query);
```

In the above example, `defination + 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.

```
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.

```
InputHandler inputHandler = executionPlanRuntime.getInputHandler("cseEventStream");
```

Use the following code to start the execution plan runtime and send events.

```

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.

For code examples, see [quick start samples for Siddhi in Github](#).

Interactive Analytics

Following sections describe how you can perform interactive analytics using the Apache Lucene Query Language in WSO2 DAS.

- [Persisting Data for Interactive Analytics](#)
- [Configuring Indexes](#)
- [Data Explorer](#)
- [Activity Explorer](#)
- [Query Language Reference](#)

Persisting Data for Interactive Analytics

After creating an event stream you can persist it by creating a corresponding table in the WSO2 Data Access Layer. Follow the steps below to persist an event stream.

1. Log in to the management console using the following URL: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Main**, and then click **Streams**.
3. Click **Edit** of the corresponding event stream which you want to persist.
4. Click **Next [Persist Event]**.
5. Select **Persist Event Stream**, and select EVENT_STORE for Record Store.
6. For each of the attribute types, do the following as required to define the schema of the event stream as shown below.

Payload Data Attributes					
Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
<input checked="" type="checkbox"/>	age	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	name	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes
No arbitrary data attributes are defined

Attribute Name : Attribute Type : Primary Key : Index Column : Score Param : Add

Advanced » Merge with existing schema (if any)

Back Save Event Stream

- Select **Persist Attribute**, if you want to persist a particular attribute.
 - Select **Primary Key**, to define an attribute type as a primary key.
 - Select **Index Column**, to enable an attribute type to be applied in searches.
 - Select **Score Param**, to define an attribute as a score parameter. For more information on score parameters, see [Searching for Data](#).
 - Select the **Is a Facet** check box if you want to persist an attribute as a facet. For more information on facets, see [Searching for Data](#).
 - Define and add any **Arbitrary Data Attributes** which you want to persist.
7. Select the **Merge with existing schema (if any)** check box if you want to persist any changes manually added to the schema of the event stream after it was created so that they would be available after the server is restarted.
 8. Click **Save Event Stream** to persist the event stream.

Configuring Indexes

WSO2 DAS has a distributed indexing engine which is built on top of Apache Lucene. Data is indexed and saved in the Data Access Layer of DAS in a store referred to as the Analytics File System. This section explains how indexing is used in WSO2 DAS.

Selecting data to be indexed

The following needs to be done when persisting data for interactive analytics.

- If you want to [search for data by a specific attribute](#) in an event stream, the **Index Column** check box should be selected for it.
- If you want to carry out faceted extended searches using a specific attribute, the **Is a Facet** check box should be selected for it.

Indexing data

The following needs to be done in order to ensure that the data is indexed as required.

- If you want to [search for data by a specific attribute](#) in an event stream, one or more events with values for the relevant attribute should be published.
- If you want to carry out faceted extended searches using a specific attribute, the value for that attribute should be entered as follows.
 - The complete value should be within square brackets.
 - The different categories embedded within the value should be separated by commas.

The following table summarizes the indexing actions required for each search action.

	Persisting Data for Analytics		Publishing Events		
Search Action	Define Attribute as Index Column?	Define Attribute as Facet?	Insert Attribute Value Within Square Brackets?	Separate Sub Values of Attribute Value with Commas?	Insert Sub Values Within Single Quotation Marks?
Searching by an attribute	Required	Not required	Not required	Not required	Required if the attribute value contains spaces.
Extracting sub categories	Required	Required	Required	Required	Required if the attribute value contains spaces.

Performing a drill down search	Required	Required	Required	Required	Required if the attribute value contains spaces.
Searching within a value range	Required	Not required	Not required	Not required	Not required

For detailed instructions to carry out different search actions using examples, see [Searching for Data](#).

Removing index data

The index data stored in your system can be cleared using one of the following methods.

Index data cannot be deleted by clearing databases.

- Clear index data via REST API. For more information, see [Clear Index Information of a Given Table via REST API](#).
- Delete the <DAS_HOME>/repository/data directory.

If you want to remove both event data and index data, see [Configuring Data Persistence - Removing persisted data](#).

Searching for Data

The following sections explain the different types of searches that can be carried out to retrieve data persisted in WSO2 DAS.

- [Searching Data Within a Time Range](#)
- [Searching Data by Primary Key Combinations](#)
- [Searching by Attributes](#)
- [Extracting Sub Categories and Performing a Drill Down Search](#)
- [Searching Data Within a Value Range](#)
- [Performing Score Functions](#)

Searching Data Within a Time Range

The following topics explain how to search for all the events in an event table that were generated during a specified time period, and how to persist data and generate events in a manner that allows such search operations to be carried out.

- [Defining and persisting attributes](#)
- [Generating data](#)
- [Searching for Data](#)

Defining and persisting attributes

The events are filtered by the time range based on their values for the `_timestamp` attribute. This attribute is added to each event table by default, and the default value for this attribute in each event is the system time at which the event was generated. If you want to override the system time, define an attribute named `_timestamp` for the required event stream as shown in the example below. This attribute should be defined as a payload attribute, and its data type should be `long`. This allows a custom timestamp to be specified for each event handled by the event stream.

[Home](#)[Help](#)

Define New Event Stream

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	<input type="text" value="BookStore"/> <small>⑦ Name of the Event Stream</small>
Event Stream Version*	<input type="text" value="1.0.0"/> <small>⑦ Version of the event stream (Eg : 1.0.0)</small>
Event Stream Description	<input type="text"/>
Event Stream Nick-Name	<input type="text"/> <small>⑦ Nick name of the event stream</small>

Stream Attributes

Meta Data Attributes

Attribute Name	Attribute Type	Actions
Transaction_ID	string	Delete

Attribute Name : Attribute Type :

Correlation Data Attributes

Attribute Name	Attribute Type	Actions
Title	string	Delete
Author	string	Delete
Price	float	Delete
Quantity	int	Delete
Date	string	Delete
Discount	float	Delete

Attribute Name : Attribute Type :

Payload Data Attributes

Attribute Name	Attribute Type	Actions
Value	float	Delete
_timestamp	long	Delete

Attribute Name : Attribute Type :

Generating data

If you are not overriding the timestamp assigned by default, there are no specific approaches to follow when generating data in order to search by time range.

If you are overriding the timestamp assigned by default, you need to specify values of the long type the _timestamp attribute you added to the event stream.

e.g., If you want to add the timestamp for an event as 16-07-1997 19:20:30, convert it to a long value (in this example, 869080830000) and enter as shown below.

[Home > List](#) [Help](#)

Event Stream Simulator

Send single event

Event Stream Name *

Stream Attributes	
Meta Attributes	
Transaction_ID(string) *	<input type="text" value="00001"/>
Correlation Attributes	
Title(string) *	<input type="text" value="Book Thief"/>
Author(string) *	<input type="text" value="Markus,Zusak"/>
Price(float) *	<input type="text" value="12.99"/>
Quantity(int) *	<input type="text" value="20"/>
Date(string) *	<input type="text" value="1997-07-16"/>
Discount(float) *	<input type="text" value="0"/>
Payload Attributes	
Value(float) *	<input type="text" value="259.80"/>
_timestamp(long) *	<input type="text" value="869080830000"/>

Searching for Data

Click on the relevant tab to search for events by time range using the DAS Management Console or via REST API.

[Via Management Console](#) [Via REST API](#)

Follow the procedure below to search for all the events in an event stream that occurred during a specific time period.

1. In the WSO2 DAS Management Console, go to the **Main** tab and click **Data Explorer** to open the **Data Explorer** page.
2. In the **Table Name** field, select the required event table.
3. Select the **By Date Range** option.
4. Select the starting date and time of the required time period in the **From** field.
5. Select the ending date and time of the required time period in the **To** field.
6. Click **Search**.

e.g., The events that occurred during the time period 11/01/2016 00:00:00 to 11/30/2016 00:00:00 can be searched for as shown below.

Data Explorer

Search

Table Name* Schedule Data Purging

Maximum Result Count

Search By Date Range By Primary Key By Query
From: To:

e.g., The events that occurred during the time period 11/01/2016 00:00:00 to 11/30/2016 00:00:00 can be searched using the following cURL command.

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/search -d '{"tableName": "BookStore", "query": "_timestamp:[1477958400000 TO 1480464000000]", "start": 0, "count": 100}' -k
```

This returns a response similar to the following.

```
[{"id": "00837303-2c04-3075-aca3-f5e702cc4574", "tableName": "BOOKSTORE", "timestamp": 1478160426000, "values": {"meta_Transaction_ID": "BSP0000008", "correlation_Title": "A Study in Scarlet", "correlation_Date": "2016,11,03", "Value": 75.0, "correlation_Author": "Arthur Conan Doyle", "correlation_Price": 5.0, "correlation_Discount": 0.25, "meta_Purchase_Batch_ID": "P000008", "correlation_Quantity": 15, "version": "1.0.0"}, {"id": "40f81b74-ef7d-3de6-a776-d74c2c1aa59d", "tableName": "BOOKSTORE", "timestamp": 1478160426000, "values": {"meta_Transaction_ID": "BSP0000008", "correlation_Title": "A Study in Scarlet", "correlation_Date": "2016,11,03", "Value": 75.0, "correlation_Author": "Arthur Conan Doyle", "correlation_Price": 5.0, "correlation_Discount": 0.25, "correlation_Quantity": 15, "version": "1.0.0"}, {"id": "5247a7f0-bd41-3270-b4b7-51a7edc9ddf3", "tableName": "BOOKSTORE", "timestamp": 1478150269000, "values": {"meta_Transaction_ID": "BP0000017", "correlation_Title": "Kim", "correlation_Date": "2016.11.03", "Value": 66.0, "correlation_Author": "Rudyard Kipling", "correlation_Price": 5.5, "correlation_Discount": 0.25, "meta_Purchase_Batch_ID": "P000017", "correlation_Quantity": 12, "version": "1.0.0"}, {"id": "c22b9285-3669-38c3-9e86-2cd2086c5f02", "tableName": "BOOKSTORE", "timestamp": 1478160425000, "values": {"meta_Transaction_ID": "BSP0000001", "correlation_Title": "The Invisible Man: A Grotesque Romance", "correlation_Date": "2016,11,03", "Value": 79.92, "correlation_Author": "Arthur C. Clarke", "correlation_Price": 6.66, "correlation_Discount": 0.0, "correlation_Quantity": 12, "version": "1.0.0"}, {"id": "9f614e49-6229-3457-9f0a-6a7edcfb76c3", "tableName": "BOOKSTORE", "timestamp": 1478160425000, "values": {"meta_Transaction_ID": "BSP0000001", "correlation_Title": "The Invisible Man: A Grotesque Romance", "correlation_Date": "2016,11,03", "Value": 79.92, "correlation_Author": "Arthur C. Clarke", "correlation_Price": 6.66, "correlation_Discount": 0.0, "meta_Purchase_Batch_ID": "P000007", "correlation_Quantity": 12, "version": "1.0.0"}}
```

Searching Data by Primary Key Combinations

The following sections explain how to search for events in an event table that match a single primary key or a combination of primary keys, and how to persist data and generate events in a manner that allows such search

operations to be carried out.

At any given time, this search retrieves only one record because each combination of primary key values is unique to one event.

The following sections explain how data needs to be persisted, generated and queried for in order to perform this type of search.

- Persisting attributes
- Generating data
- Searching for Data

Persisting attributes

In order to perform this search, you should select the **Primary Key** check box for attributes to be considered primary keys at the time of persisting data for the relevant event stream.

Home Help

Edit Event Stream

Enter Event Stream Details

Persist Event Stream

Record Store
EVENT_STORE (edit)

Meta Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Transaction_ID	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Purchase_Batch_ID	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Correlation Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Title	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Author	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Price	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Quantity	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Date	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Discount	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Payload Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Value	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	_timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes

No arbitrary data attributes are defined

Attribute Name :	Attribute Type :	Primary Key :
<input type="text"/>	INTEGER	<input type="checkbox"/>
Index Column :	<input type="checkbox"/>	Score Param :
Is a Facet :	<input type="checkbox"/>	Add

Advanced (edit)

Back Save Event Stream

Generating data

- When there is only one attribute persisted as a primary key, the value specified for that attribute should be unique for each event.
- When there are two or more attributes persisted as primary keys, the value specified for an individual primary key does not have to be unique for each event. However, each event should have a unique combination of primary key values.

e.g., If an event stream has only one primary key named `Transaction_ID`, a unique transaction ID should be specified for each event generated. If you generate multiple events with the same transaction ID, the only the last event generated with that transaction ID is persisted. However, if there are two primary keys named `Transaction_ID` and `Purchase_Batch_ID`, two or more events can have the same transaction ID as long as their purchase batch IDs are different to each other.

Searching for Data

Click on the relevant tab to search for events by primary keys using the DAS Management Console or via REST API.

Via Management Console/Via REST API

Follow the procedure below to search for all the events in an event stream by a single or a combination of primary keys.

1. In the WSO2 DAS Management Console, go to the **Main** tab and click **Data Explorer** to open the **Data Explorer** page.
2. In the **Table Name** field, select the required event table.
3. Select the **Primary Key** option. The fields for the available primary keys in the stream appear as shown below.

This option is not visible when the event table you selected does not have any attributes defined as primary keys.

4. Enter the required values in the relevant primary key fields, and click **Search**.

If you do not enter values for one or more primary key fields displayed, the system searches for events with no values for those primary key attributes.

e.g., If there are two primary keys named `Transaction_ID` and `Purchase_Batch_ID`, and you can search for events with transaction ID `BSP000001` without specifying a purchase batch ID, the system searches for an event with `BSP000001` as the transaction ID and no purchase batch ID as shown below.

Data Explorer

Search

Table Name* **BOOKSTORE**

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

meta_Purchase_Batch_ID
meta_Transaction_ID **BSP0000001**

Results

Total Records: 0

BOOKSTORE									
meta_Transaction_ID	meta_Purchase_Batch_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp
BSP0000001		The Invisible Man: A Grotesque Romance	Arthur,C,Clarke	6.66	12	2016,11,03	0.0	79.92	2016-11-03 13:37:05 IST

Row count: 10

If you enter values for both primary key fields, the system searches for a record that matches the given combination for those two fields as shown below.

Home > Manage > Interactive Analytics > Data Explorer

[Help](#)

Data Explorer

Search

Table Name* **BOOKSTORE**

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

meta_Purchase_Batch_ID **P000007**
meta_Transaction_ID **BSP0000001**

Results

Total Records: 0

BOOKSTORE									
meta_Transaction_ID	meta_Purchase_Batch_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp
BSP0000001	P000007	The Invisible Man: A Grotesque Romance	Arthur,C,Clarke	6.66	12	2016,11,03	0.0	79.92	2016-11-03 13:37:05 IST

Row count: 10

e.g., The following cURL command can be used to search for the event with **BSP0000001** as the transaction ID and no purchase batch ID.

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/tables/BookStore/keyed_records -d '{ "valueBatches" : [ { "meta_Transaction_ID" : "BSP0000001", "meta_purchase_batch_ID" : " " } ] , "columns" : [ "correlation_Title", "correlation_Author", "correlation_price" ] }' -k
```

It returns a response similar to the following.

```
[ { "id" : "c22b9285-3669-38c3-9e86-2cd2086c5f02", "tableName" : "BOOKSTORE", "timestamp" : 1478160425000, "values" : { "meta_Transaction_ID" : "BSP0000001", "correlation_Title" : "The Invisible Man: A Grotesque Romance", "correlation_Date" : "2016,11,03", "Value" : 79.92, "correlation_Author" : "Arthur,C,Clarke" } }
```

```
,Clarke","correlation_Price":6.66,"correlation_Discount":0.0,"correlation_Quantity":1
2,"_version":"1.0.0"}]
```

The following cURL command can be used to search for the event with BSP000001 as the transaction ID, and P000007 as the purchase batch ID.

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic
YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/tables/BookStore/keyed_records
-d '{"valueBatches": [{"meta_Transaction_ID": "BSP000001", "meta_purchase_batch_ID": ""}, {"meta_Transaction_ID": "BSP000001", "meta_purchase_batch_ID": "P000007"}], "columns": ["correlation_Title", "correlation_Author", "correlation_price"]}' -k
```

It returns a response similar to the following.

```
[{"id": "c22b9285-3669-38c3-9e86-2cd2086c5f02", "tableName": "BOOKSTORE", "timestamp": 147
8160425000, "values": {"meta_Transaction_ID": "BSP000001", "correlation_Title": "The
I n v i s i b l e M a n : A G r o t e s q u e
Romance", "correlation_Date": "2016,11,03", "Value": 79.92, "correlation_Author": "Arthur,C
,Clarke", "correlation_Price": 6.66, "correlation_Discount": 0.0, "correlation_Quantity": 1
2, "_version": "1.0.0"}}]
```

Searching by Attributes

The following topics explain how to search for events by one or more attributes, and how to persist data and generate events in a manner that allows such search operations to be carried out.

- [Persisting attributes](#)
- [Generating data](#)
- [Searching for Data](#)

Persisting attributes

The **Index Column** should be selected at the time of persisting the event stream for attributes by which you want to search.

[Home](#)[Help](#)

Edit Event Stream

Enter Event Stream Details

<input checked="" type="checkbox"/> Persist Event Stream

Record Store
EVENT_STORE

Meta Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Transaction_ID	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Purchase_Batch_ID	STRING	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Correlation Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Title	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Author	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Price	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Quantity	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Date	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Discount	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Payload Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Value	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	_timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes

No arbitrary data attributes are defined

Attribute Name :	Attribute Type :	Primary Key :	Index Column :	Score Param :	Is a Facet :	Add
<input type="text"/>	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

[Advanced](#)

[Back](#) [Save Event Stream](#)

Generating data

There are no specific guidelines to follow at the time of entering data in order to allow this search operation.

Searching for Data

Click on the relevant tab to search for events by attributes using the DAS Management Console or via REST API.

[Via Management Console](#) [Via REST API](#)

Follow the procedure below to search for events in an event stream by one or more attributes.

1. In the WSO2 DAS Management Console, go to the **Main** tab and click **Data Explorer** to open the **Data Explorer** page.
2. In the **Table Name** field, select the required event table.
3. Select the **By Query** option. This displays the query field as shown below.

Data Explorer

Search

Table Name*	BOOKSTORE	Schedule Data Purging
Maximum Result Count	1000	
Search	<input type="radio"/> By Date Range <input type="radio"/> By Primary Key <input checked="" type="radio"/> By Query	
Search Query correlation_Price:7.00		
Select a Facet		
Search Reset		

4. To search for records where the price is 7.00, enter a query as shown below.

`correlation_Price:7.00`

Home > Manage > Interactive Analytics > Data Explorer

Data Explorer

Search

Table Name*	BOOKSTORE	Schedule Data Purging
Maximum Result Count	1000	
Search	<input type="radio"/> By Date Range <input type="radio"/> By Primary Key <input checked="" type="radio"/> By Query	
correlation_Price:7.00		
Select a Facet		
Search Reset		

Click **Search**. The records that match the search are displayed as shown in the example below.

Results

Total Records: 6 (198 ms)

BOOKSTORE										
	meta_Transaction_ID	meta_Purchase_Batch_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp
iii	BSP0000011		Mary Called Magdalene	Margaret,George	7.0	10	2016,10,02	0.5	70.0	2017-10-02 13:37:05 IST
iii	BSP0000013		Elizabeth I	Margaret,George	7.0	10	2017,01,30	0.5	70.0	2017-01-30 13:37:05 IST
iii	BSP0000012	P000014	Memoirs of Cleopatra	Margaret,George	7.0	10	2016,08,29	0.5	70.0	2017-08-29 13:37:05 IST
iii	BSP0000013	P000011	Elizabeth I	Margaret,George	7.0	10	2017,01,30	0.5	70.0	2017-01-30 13:37:05 IST
iii	BSP0000011	P000015	Mary Called Magdalene	Margaret,George	7.0	10	2016,10,02	0.5	70.0	2017-10-02 13:37:05 IST
iii	BSP0000012		Memoirs of Cleopatra	Margaret,George	7.0	10	2016,08,29	0.5	70.0	2017-08-29 13:37:05 IST

<< < 1 > >> Go to page: 1 Row count: 10 Showing 1-6 of 6

5. To search for books of which the price is 5.50 and/or the discount is 0.25, enter a query as follows.

```
correlation_Price: 5.5
correlation_Discount: 0.25
```

Home > Manage > Interactive Analytics > Data Explorer

Data Explorer

The screenshot shows the 'Data Explorer' interface. At the top, there is a search bar labeled 'Search'. Below it, the 'Table Name*' dropdown is set to 'BOOKSTORE', and the 'Maximum Result Count' dropdown is set to '1000'. Under the 'Search' section, there are three radio button options: 'By Date Range', 'By Primary Key', and 'By Query'. The 'By Query' option is selected, and the query entered is 'correlation_Price:5.5 correlation_Discount:0.25'. Below the query input, there is a 'Select a Facet' dropdown. At the bottom of the search panel are 'Search' and 'Reset' buttons.

Click **Search**. As shown in the example below, all the records where the price is 5.50 and/or the discount is 0.25 are displayed.

Results										
Total Records: 9 (130 ms)										
BOOKSTORE										
meta_Transaction_ID	meta_Purchase_Batch_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp	
BPO000017	P000017	Kim	Rudyard,Kipling	5.5	12	2016.11.03	0.25	66.0	2016-11-03 10:47:49 IST	
BSP0000002		Childhood's End	Arthur,C,Clarke	5.5	12	2016.10.06	0.0	66.0	2016-10-06 13:37:05 IST	
BSP0000002	P000003	Childhood's End	Arthur,C,Clarke	5.5	12	2016.10.06	0.0	66.0	2016-10-06 13:37:05 IST	
BSP0000007		The Adventures of Sherlock Holmes	Arthur,Conan,Doyle	6.0	15	2017.01.30	0.25	90.0	2017-01-30 13:37:07 IST	
BSP0000009	P000004	Sherlock Holmes: The Complete Novels and Stories	Arthur,Conan,Doyle	9.5	30	2016.10.06	0.25	285.0	2016-10-06 13:37:06 IST	
BSP0000009		Sherlock Holmes: The Complete Novels and Stories	Arthur,Conan,Doyle	9.5	30	2016.10.06	0.25	285.0	2016-10-06 13:37:06 IST	
BSP0000008	P000008	A Study in Scarlet	Arthur,Conan,Doyle	5.0	15	2016.11.03	0.25	75.0	2016-11-03 13:37:06 IST	
BSP0000008		A Study in Scarlet	Arthur,Conan,Doyle	5.0	15	2016.11.03	0.25	75.0	2016-11-03 13:37:06 IST	
BSP0000007	P000012	The Adventures of Sherlock Holmes	Arthur,Conan,Doyle	6.0	15	2017.01.30	0.25	90.0	2017-01-30 13:37:07 IST	

<< < 1 > >> Go to page: 1 Row count: 10 Showing 1-9 of 9

e.g., To search for records where the price is 7.00, issue the following cURL command.

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/search -d '{"tableName": "BookStore", "query": "correlation_Price:7.0", "start": 0, "count": 100}' -k
```

It generates a response similar to the following.

```
[{"id": "53dba6d7-2ed2-3710-9c98-2a20f0a71766", "tableName": "BOOKSTORE", "timestamp": 1506931625000, "values": {"meta_Transaction_ID": "BSP0000011", "correlation_Title": "Mary C a l l e d Magdalene", "correlation_Date": "2016,10,02", "Value": 70.0, "correlation_Author": "Margaret,George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "correlation_Quantity": 10, "_version": "1.0.0"}}, {"id": "7959f493-34f0-3755-84ea-92a96101c9a4", "tableName": "BOOKSTORE", "timestamp": 1485763625000, "values": {"meta_Transaction_ID": "BSP0000013", "correlation_Title": "Elizabeth I", "correlation_Date": "2017,01,30", "Value": 70.0, "correlation_Author": "Margaret,George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "correlation_Quantity": 10, "_vers
```

```

ion": "1.0.0"}}, {"id": "f9011238-7699-3133-af50-d5ae37ef2f2c", "tableName": "BOOKSTORE", "timestamp": 1503994025000, "values": {"meta_Transaction_ID": "BSP0000012", "correlation_Title": "Memoirs of Cleopatra", "correlation_Date": "2016,08,29", "Value": 70.0, "correlation_Author": "Margaret, George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "meta_Purchase_Batch_ID": "P000014", "correlation_Quantity": 10, "_version": "1.0.0"}}, {"id": "2195bd0c-daa8-3d3f-952f-bc75f3cc39d9", "tableName": "BOOKSTORE", "timestamp": 1485763625000, "values": {"meta_Transaction_ID": "BSP0000013", "correlation_Title": "Elizabeth I", "correlation_Date": "2017,01,30", "Value": 70.0, "correlation_Author": "Margaret, George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "meta_Purchase_Batch_ID": "P000011", "correlation_Quantity": 10, "_version": "1.0.0"}}, {"id": "cea98647-3155-3742-977e-1992c9d1964b", "tableName": "BOOKSTORE", "timestamp": 1506931625000, "values": {"meta_Transaction_ID": "BSP0000011", "correlation_Title": "Mary Magdalene", "correlation_Date": "2016,10,02", "Value": 70.0, "correlation_Author": "Margaret, George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "meta_Purchase_Batch_ID": "P000015", "correlation_Quantity": 10, "_version": "1.0.0"}}, {"id": "281e4ce2-0051-3clf-8924-0533511a1b0c", "tableName": "BOOKSTORE", "timestamp": 1503994025000, "values": {"meta_Transaction_ID": "BSP0000012", "correlation_Title": "Memoirs of Cleopatra", "correlation_Date": "2016,08,29", "Value": 70.0, "correlation_Author": "Margaret, George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "correlation_Quantity": 10, "_version": "1.0.0"}}

```

Extracting Sub Categories and Performing a Drill Down Search

The following topics explain how to carry out faceted searches by extracting sub categories of one or more attributes, and how to persist data and generate events in a manner that allows such search operations to be carried out.

- Persisting attributes
- Generating data
- Searching for Data

Persisting attributes

In order carry out faceted searches for an attribute, the **Is a Facet** check box should be selected for that attribute. In the example below, this check box is selected for the **Author** and **Date** attributes are persisted as facets.

Home

[Help](#)

Edit Event Stream

Enter Event Stream Details

Persist Event Stream

Record Store
EVENT_STORE

Meta Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Transaction_ID	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Purchase_Batch_ID	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Correlation Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Title	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Author	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Price	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Quantity	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Date	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Discount	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Payload Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Value	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	_timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes

No arbitrary data attributes are defined

Attribute Name :
Attribute Type :

Primary Key : Index Column : Score Param : Is a Facet :

[Advanced](#)

Generating data

In order to separate subcategories, the values for attributes persisted as facets should be specified as comma separated values.

In the following example, the comma (,) separated values entered for the **Author** attribute define a hierarchy of categories when searching by the Author facet. The first name is the main category, the second name is a sub category of the first name, etc.

[Home > List](#) [Help](#)

Event Stream Simulator

Send single event

Event Stream Name *

Stream Attributes	
Meta Attributes	
Transaction_ID(string) *	<input type="text" value="BSP0000015"/>
Purchase_Batch_ID(string) *	<input type="text" value="P0000020"/>
Correlation Attributes	
Title(string) *	<input type="text" value="The Old Curiosity Shop"/>
Author(string) *	<input type="text" value="Charles,Dickens"/>
Price(float) *	<input type="text" value="4.95"/>
Quantity(int) *	<input type="text" value="20"/>
Date(string) *	<input type="text" value="2016,11,16"/>
Discount(float) *	<input type="text" value="0"/>
Payload Attributes	
Value(float) *	<input type="text" value="99"/>
_timestamp(long) *	<input type="text" value="1479299798000"/>

Searching for Data

Click on the relevant tab to extract sub categories and perform drill down searches using the DAS Management Console or via REST API.

[Via Management Console](#) [Via REST API](#)

Follow the procedure below to search for events in an event stream by one or more attributes.

1. In the WSO2 DAS Management Console, go to the **Main** tab and click **Data Explorer** to open the **Data Explorer** page.
2. In the **Table Name** field, select the required event table.
3. Select the **By Query** option. This displays the query field as shown below.

Data Explorer

Search

Table Name*	BOOK_STORE	Schedule Data Purging
Maximum Result Count	1000	
Search	<input type="radio"/> By Date Range <input checked="" type="radio"/> By Query	
Search Query		
Select a Facet		
<input type="button" value="Search"/> <input type="button" value="Reset"/>		

4. To search for books by a specific faceted attribute (e.g., author), select that attribute in the **Select a Facet** field.

Data Explorer

Search

Table Name*	BOOKSTORE	Schedule Data Purging
Maximum Result Count	1000	
Search	<input type="radio"/> By Date Range <input type="radio"/> By Primary Key <input checked="" type="radio"/> By Query	
Search Query		
correlation_Author		
correlation_Author <input type="button" value="Select a category"/> <input type="button" value="Remove"/>		
<input type="button" value="Search"/> <input type="button" value="Reset"/>		

As a result, a separate field is added for the attribute (the **correlation_Author** field in the above example).

5. In the new field added for the facet, select a value as demonstrated in the example below. The value you select represents the main category (in this example, the first name of the author).

Data Explorer

Search

Table Name*	BOOKSTORE	Schedule Data Purging
Maximum Result Count	1000	
Search	<input type="radio"/> By Date Range <input type="radio"/> By Primary Key <input checked="" type="radio"/> By Query	
Search Query		
correlation_Author		
correlation_Author Arthur <input type="button" value="Select a category"/> <input type="button" value="Remove"/>		
<input type="button" value="Search"/> <input type="button" value="Reset"/>		

This adds a second field for the selected facet as shown above.

6. To extract the sub categories available for the category you selected, expand the second field as

demonstrated below. In this example, it shows all the available authors whose first name is Arthur.

The screenshot shows the WSO2 Data Explorer search interface. The search query is set to "correlation_Author" and the value is "Arthur". A dropdown menu is open, showing categories: "Select a category", "Conan", "C", and "Golden". The "Select a category" option is checked. The "Search" and "Reset" buttons are at the bottom left.

- Select one sub category and click **Search**. This carries out a drill down search and retrieves records that match both the category and the sub category.

The screenshot shows the WSO2 Data Explorer search interface. The search query is set to "correlation_Author" and the value is "Arthur". The dropdown menu is open, showing categories: "Select a category", "Conan", and "Golden". The "Conan" option is selected. The "Search" button is highlighted with a red box. The "Reset" button is at the bottom left.

In this example, records where the first name of the author is Arthur and the second name is Conan are retrieved.

BOOKSTORE										
meta_Transaction_ID	meta_Purchase_Batch_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp	
BSP0000007	P000012	The Adventures of Sherlock Holmes	Arthur,Conan,Doyle	6.0	15	2017,01,30	0.25	90.0	2017-01-30 13:37:07 IST	
BSP0000008		A Study in Scarlet	Arthur,Conan,Doyle	5.0	15	2016,11,03	0.25	75.0	2016-11-03 13:37:06 IST	
BSP0000008	P000008	A Study in Scarlet	Arthur,Conan,Doyle	5.0	15	2016,11,03	0.25	75.0	2016-11-03 13:37:06 IST	
BSP0000009		Sherlock Holmes: The Complete Novels and Stories	Arthur,Conan,Doyle	9.5	30	2016,10,06	0.25	285.0	2016-10-06 13:37:06 IST	
BSP0000009	P000004	Sherlock Holmes: The Complete Novels and Stories	Arthur,Conan,Doyle	9.5	30	2016,10,06	0.25	285.0	2016-10-06 13:37:06 IST	
BSP0000007		The Adventures of Sherlock Holmes	Arthur,Conan,Doyle	6.0	15	2017,01,30	0.25	90.0	2017-01-30 13:37:07 IST	

- If you want to search by two facets, select another facet as shown in the example below.

Data Explorer

Search

Table Name* **BOOKSTORE** Schedule Data Purging

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

Search Query

correlation_Date

correlation_Author Arthur Conan Select a category Remove

correlation_Date Select a category Remove

Search **Reset**

9. Select categories and sub categories for both facets as shown in the example below, and click **Search**.

Data Explorer

Search

Table Name* **BOOKSTORE** Schedule Data Purging

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

Search Query

correlation_Date

correlation_Author Arthur Conan Doyle Select a category Remove

correlation_Date 2016 11 03 Select a category Remove

Search **Reset**

This retrieves records that match the criteria specified by both facets as shown below.

Results										
Total Records: 2 (77 ms)										
BOOKSTORE										
	meta_Transaction_ID	meta_Purchase_Batch_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp
III	BSP0000008		A Study in Scarlet	Arthur,Conan,Doyle	5.0	15	2016,11,03	0.25	75.0	2016-11-03 13:37:06 IST
III	BSP0000008	P000008	A Study in Scarlet	Arthur,Conan,Doyle	5.0	15	2016,11,03	0.25	75.0	2016-11-03 13:37:06 IST

<< < 1 >> Go to page: 1 Row count: 10 Showing 1-2 of 2

e.g., The following cURL command can be used to extract the first names of the available authors.

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRTaw4=" -v https://localhost:9443/analytics/facets -d '{"tableName' : "BookStore", "fieldName" : "correlation_Author", "categoryPath" : [], "query" : "*:*", "scoreFunction" : "1"}' -k
```

This returns a response similar to the following.

```
{"categoryPath":[], "categoryCount":7, "categories": {"Arthur":12.0, "Margaret":8.0, "Charles":4.0, "Jane":2.0, "George":2.0, "Anthony":2.0, "Rudyard":1.0}}
```

Searching Data Within a Value Range

The following topics explain how to search for records with values within a specific range for a selected attribute, and how to persist data in a manner that allows such search operations to be carried out.

- Persisting attributes
- Generating data
- Searching for Data

Persisting attributes

Data can be searched by value range only for attributes with numeric values. Therefore, the data type of the attribute by which the search is carried out should be `INTEGER` or `FLOAT`. The attribute should also be persisted as an index column as shown below.

The screenshot shows the 'Edit Event Stream' page in WSO2 Data Analytics Server. The 'Persist Event Stream' checkbox is checked. The 'Record Store' dropdown is set to 'EVENT_STORE'. The 'Meta Data Attributes' section contains two rows: 'Transaction_ID' (STRING) and 'Purchase_Batch_ID' (STRING). The 'Correlation Data Attributes' section contains five rows: 'Title' (STRING), 'Author' (STRING), 'Price' (FLOAT, highlighted with a red border), 'Quantity' (INTEGER), 'Date' (STRING), and 'Discount' (FLOAT). The 'Payload Data Attributes' section contains two rows: 'Value' (FLOAT) and '_timestamp' (LONG). The 'Arbitrary Data Attributes' section has a note: 'No arbitrary data attributes are defined'. Below it is an 'Add' button with fields for Attribute Name (empty), Attribute Type (INTEGER), Primary Key (unchecked), Index Column (unchecked), Score Param (unchecked), and Is a Facet (unchecked). The 'Advanced' tab is selected. At the bottom are 'Back' and 'Save Event Stream' buttons.

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input type="checkbox"/>	Transaction_ID	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	Purchase_Batch_ID	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input type="checkbox"/>	Title	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	Author	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Price	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	Quantity	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	Date	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	Discount	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input type="checkbox"/>	Value	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	_timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes

No arbitrary data attributes are defined

Attribute Name :
Attribute Type :
Primary Key : Index Column : Score Param : Is a Facet :

Advanced

Generating data

There are no specific guidelines to follow at the time of entering data in order to allow this search operation.

Searching for Data

e.g., The following cURL command can be used to search for events of which the price is between 5.00 and 7.00.

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/search -d '{"tableName": "BookStore", "query": "correlation_Price:[5.00 TO 7.00]", "start": 0, "count": 100}' -k
```

It returns a response similar to the following.

```
[{"id": "1727106f-5cd9-3249-8328-b952a0bcef27", "tableName": "BOOKSTORE", "timestamp": 1485763627000, "values": {"meta_Transaction_ID": "BSP0000007", "correlation_Title": "The Adventures of Sherlock Holmes", "correlation_Date": "2017,01,30", "Value": 90.0, "correlation_Author": "Arthur Conan, Doyle", "correlation_Price": 6.0, "correlation_Discount": 0.25, "correlation_Quantity": 15, "_version": "1.0.0"}, {"id": "53dba6d7-2ed2-3710-9c98-2a20f0a71766", "tableName": "BOOKSTORE", "timestamp": 1506931625000, "values": {"meta_Transaction_ID": "BSP0000011", "correlation_Title": "Mary Magdalene", "correlation_Date": "2016,10,02", "Value": 70.0, "correlation_Author": "Margaret, George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "correlation_Quantity": 10, "_version": "1.0.0"}, {"id": "af3d4a10-0c7b-34b1-b436-793368d2877d", "tableName": "BOOKSTORE", "timestamp": 1475741225000, "values": {"meta_Transaction_ID": "BSP0000002", "correlation_Title": "Childhood's End", "correlation_Date": "2016,10,06", "Value": 66.0, "correlation_Author": "Arthur, C, Clarke", "correlation_Price": 5.5, "correlation_Discount": 0.0, "correlation_Quantity": 12, "_version": "1.0.0"}, {"id": "7959f493-34f0-3755-84ea-92a96101c9a4", "tableName": "BOOKSTORE", "timestamp": 1485763625000, "values": {"meta_Transaction_ID": "BSP0000013", "correlation_Title": "Elizabeth I", "correlation_Date": "2017,01,30", "Value": 70.0, "correlation_Author": "Margaret, George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "correlation_Quantity": 10, "_version": "1.0.0"}, {"id": "3c251430-c915-3c8c-9c03-6dc3364865aa", "tableName": "BOOKSTORE", "timestamp": 1483344425000, "values": {"meta_Transaction_ID": "BSP0000015", "correlation_Title": "1984", "correlation_Date": "2017,01,02", "Value": 139.8, "correlation_Author": "George, Orwell", "correlation_Price": 6.99, "correlation_Discount": 0.0, "correlation_Quantity": 20, "_version": "1.0.0"}, {"id": "5254de6d-6df0-3d82-b3a5-d1aba76adb18", "tableName": "BOOKSTORE", "timestamp": 1475741225000, "values": {"meta_Transaction_ID": "BSP0000002", "correlation_Title": "Childhood's End", "correlation_Date": "2016,10,06", "Value": 66.0, "correlation_Author": "Arthur, C, Clarke", "correlation_Price": 5.5, "correlation_Discount": 0.0, "meta_Purchase_Batch_ID": "P000003", "correlation_Quantity": 12, "_version": "1.0.0"}, {"id": "00837303-2c04-3075-aca3-f5e702cc4574", "tableName": "BOOKSTORE", "timestamp": 1478160426000, "values": {"meta_Transaction_ID": "BSP0000008", "correlation_Title": "A Study in Scarlet", "correlation_Date": "2016,11,03", "Value": 75.0, "correlation_Author": "Arthur, Conan, Doyle", "correlation_Price": 5.0, "correlation_Discount": 0.25, "meta_Purchase_Batch_ID": "P000008", "correlation_Quantity": 15, "_version": "1.0.0"}, {"id": "09ce7320-4795-3461-ad8e-69e6dfc45f73", "tableName": "BOOKSTORE", "timestamp": 1472717226000, "values": {"meta_Transaction_ID": "BSP0000010", "correlation_Title": "Gone With the Wind", "correlation_Date": "2016,09,01", "Value": 136.6, "correlation_Author": "Margaret, Mitchell", "correlation_Price": 6.83, "correlation_Discount": 0.0, "correlation_Quantity": 20, "_version": "1.0.0"}, {"id": "f9011238-7699-3133-af50-d5ae37ef2f2c", "tableName": "BOOKSTORE", "timestamp": 1503994025000, "values": {"meta_Transaction_ID": "BSP0000012", "correlation_Title": "Memoirs of Cleopatra", "correlation_Date": "2016,08,29", "Value": 70.0, "correlation_Author": "Margaret, George", "correlation_Price": 7.0, "correlation_Discount": 0.5, "meta_Purchase_Batch_ID": "P000014", "correlation_Quantity": 10, "_version": "1.0.0"}, {"id": "489f17c8-884f-3f0f-80a9-29cf06e660d7", "tableName": "BOOKSTORE", "timestamp": 1483603625000, "values": {"meta_Transaction_ID": "BSP0000004", "correlation_Title": "Emma", "correlation_Date": "2017,01,05", "Value": 134.6, "correlation_Author": "Jane, Austen", "correlation_Price": 6.73, "correlation_Discount": 0.0, "correlation_Quantity": 20, "_version": "1.0.0"}, {"id": "4b68cbf0-51ea-3388-844b-bfa7847c9ffa", "tableName": "BOOKSTORE", "timestamp": 1477901225000, "values": {"meta_Transaction_ID": "BSP0000006", "correlation_Title": "The Origin of
```



```

ty":10,"_version":"1.0.0"}],{"id":"9f614e49-6229-3457-9f0a-6a7edcfb76c3","tableName":
"BOOKSTORE","timestamp":1478160425000,"values":{"meta_Transaction_ID":"BSP0000001","c
orrelation_Title":"The Invisible Man: A Grotesque
Romance","correlation_Date":"2016,11,03","Value":79.92,"correlation_Author":"Arthur,C
,Clarke","correlation_Price":6.66,"correlation_Discount":0.0,"meta_Purchase_Batch_ID"
:"P000007","correlation_Quantity":12,"_version":"1.0.0"}}, {"id":"c9b6d1f8-561d-3258-b
3eb-8ddd63f135c7","tableName":"BOOKSTORE","timestamp":1485763627000,"values":{"meta_T
ransaction_ID":"BSP0000007","correlation_Title":"The Adventures of Sherlock
Holmes","correlation_Date":"2017,01,30","Value":90.0,"correlation_Author":"Arthur,Con
an,Doyle","correlation_Price":6.0,"correlation_Discount":0.25,"meta_Purchase_Batch_ID"
:"P000012","correlation_Quantity":15,"_version":"1.0.0"}}, {"id":"7dde0cf5-bd42-3df3-
add7-6a91b3cbleb4","tableName":"BOOKSTORE","timestamp":1472717226000,"values":{"meta_
Transaction_ID":"BSP0000010","correlation_Title":"Gone With the
Wind","correlation_Date":"2016,09,01","Value":136.6,"correlation_Author":"Margaret,Mi
tchell","correlation_Price":6.83,"correlation_Discount":0.0,"meta_Purchase_Batch_ID":"
P000002","correlation_Quantity":20,"_version":"1.0.0"}]

```

Performing Score Functions

The following sections explain how to perform score functions for specific attributes when carrying out searches, and how to persist data to enable such searches to be carried out.

- Persisting attributes
- Generating Data
- Searching for data

Persisting attributes

The attributes to be used for score functions should be persisted as index columns and score parameters at the time of persisting attributes. In the example given below, the **Price** and **Discount** attributes are persisted for the purpose of performing score functions.

Edit Event Stream

Enter Event Stream Details

Persist Event Stream

Record Store
EVENT_STORE

Meta Data Attributes

<input checked="" type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Transaction_ID	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Purchase_Batch_ID	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Correlation Data Attributes

<input checked="" type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Title	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Author	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Price	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Quantity	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Date	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Discount	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Payload Data Attributes

<input checked="" type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Value	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	_timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes

No arbitrary data attributes are defined

Attribute Name :
Attribute Type :

Primary Key : Index Column : Score Param : Is a Facet :

Advanced

Generating Data

There are no specific guidelines to follow at the time of entering data in order to allow this search operation.
Searching for data

e.g., The following cURL command outputs the result os the score function Price - Discount for all the available titles.

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://100.5.73:9443/analytics/facets -d '{"tableName": "BookStore", "fieldName": "correlation_Title", "categoryPath": [], "query": "*:*", "scoreFunction": "correlation_Price * ((100 - correlation_Discount))"}' -k
```

This generates a response similar to the following.

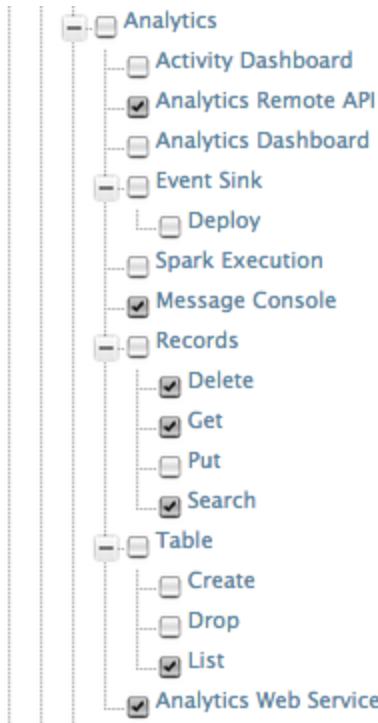
```
{"categoryPath": [], "categoryCount": 10, "categories": {"SampleTitle": 6.305843089461677E-42, "Sinatra: The Life": 2.802596928649634E-42, "Sherlock Holmes: The Complete Novels and Stories": 2.5223372357846707E-42, "The Invisible Man: A Grotesque Romance": 1.6815581571897805E-42, "The Origin of
```

```
Species":1.6815581571897805E-42,"Oliver Twist":1.6815581571897805E-42,"Gone With the Wind":1.6815581571897805E-42,"A Study in Scarlet":1.401298464324817E-42,"Childhood's End":1.401298464324817E-42,"Kim":7.006492321624085E-43}}
```

Data Explorer

Data Explorer is the single-point of user interactions, which allows you to carry out different operations related to data analytics. Data Explorer allows you to search and view tables in the **Data Access Layer (DAL)** of WSO2 DAS, and browse their records. You can authorize and restrict users on the functions that they carry out using the Data Explorer, by setting permissions on the user roles assigned to them as shown below.

For instructions on setting the following permissions, see [Adding and Managing Users and Roles](#).



Data Explorer function	Required permissions
All functions	<ul style="list-style-type: none"> Analytics->Data Explorer Analytics->Analytics Web Service Analytics->Analytics Remote API
View records of a table	<ul style="list-style-type: none"> Analytics->Table-> List Analytics->Records->Get
Search by date range	<ul style="list-style-type: none"> Analytics->Table-> List Analytics->Records->Get
Search by primary key	<ul style="list-style-type: none"> Analytics->Table-> List Analytics->Records->Get
Search by query	<ul style="list-style-type: none"> Analytics->Table-> List Analytics->Records->Search
Schedule data purging	<ul style="list-style-type: none"> Analytics->Records->Delete

The Data Explorer user functions are explained below.

- Viewing records of a table
- Searching for records of a table

Viewing records of a table

Follow the steps below to view records of existing tables using the Data Explorer.

1. Log in to the WSO2 DAS Data Explorer using the following URL: `https://<DAS_HOST>:<DAS_PORT>carbon/`
2. Click **Main**, and then click **Data Explorer** in the **Interactive Analytics** menu.
3. Select the **Table Name** of which you want to view records.
4. Click **Search**. You view the all records of the selected table as shown below.

The screenshot shows the WSO2 Data Explorer interface. At the top, there is a navigation bar with links for Home, Manage, Interactive Analytics, and Data Explorer. On the right side of the header, there is a Help link. Below the header, there is a search form with fields for 'Table Name*' (set to 'BOOK_STORE'), 'Maximum Result Count' (set to '1000'), and search options ('By Date Range', 'By Primary Key', 'By Query'). There are also 'Search' and 'Reset' buttons. The main area is titled 'Results' and contains a note: 'Note: Total record count for the table is not available.' Below this, there is a table titled 'BOOK_STORE' with columns: ID, Title, Author, Published-Date, Category, Count, and timestamp. The table lists 10 records from the BOOK_STORE table. At the bottom of the results table, there are pagination controls ('<< < 1 2 ... 99 100 > >>'), a page number selector ('Go to page: 1'), a row count selector ('Row count: 10'), and a message: 'Showing 1-10 of 1000'.

ID	Title	Author	Published-Date	Category	Count	timestamp
00001	A Tale of Two Cities	Charles Dickens	'[1659,'04','01']	Fiction,Classic,	120	2015-10-28 10:00:31 IST
00002	The Casual Vacancy	J.K.Rowling	2012,09,27	Fiction,Modern,	120	2015-10-28 10:02:10 IST
00003	Cocktail Time	P.G. Wodehouse	1958,06,12	Fiction,Humour,	120	2015-10-28 10:03:54 IST
00005	The Victorians	A.N. Wilson	1988	History,Victorian Era	200	2015-10-28 10:07:22 IST
00006	Elizabeth the Queen	Alison Weir	1998	History,Elizabethan Era	200	2015-10-28 10:09:03 IST
00007	How to Win Friends and Influence People	Dale Carnegie	1936	Self-Help,Sociology	200	2015-10-28 10:12:54 IST
00008	The Wealth of Nations	Adam Smith	1776	Economics,Classical	200	2015-10-28 10:14:51 IST

Searching for records of a table

You can browse existing tables in the WSO2 DAS and search for records by filtering them based on a date range or by filtering them using a query as explained below

Searching by a date range

Follow the steps below to search for records of an existing table based on a particular date range.

1. Log in to the WSO2 DAS management console using the following URL, if you are not already logged in: `https://<DAS_HOST>:<DAS_PORT>carbon/`
2. Click **Main**, and then click **Data Explorer** in the **Interactive Analytics** menu.
3. Select the **Table Name** of which you want to search and view records.
4. Select **By Date Range** for **Search**.
5. Enter the **From** and **To** dates to define the date range within which you want to search the records.

You can specify a time period within which you want to search and view records by selecting the start and end times in hours and minutes in 24 hours time standard.

6. Click **Search**. You view the filtered records of the selected table as shown below.

Data Explorer

Search

Table Name* **BOOK_STORE**

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query
From: **10/28/2015 09:11:10** To: **10/28/2015 10:06:01**

Results

Note: Total record count for the table is not available.

BOOK_STORE						
ID	Title	Author	Published-Date	Category	Count	timestamp
00001	A Tale of Two Cities	Charles Dickens	['1859','04','01']	Fiction,Classic,	120	2015-10-28 10:00:31 IST
00002	The Casual Vacancy	J.K.Rowling	2012,09,27	Fiction,Modern,	120	2015-10-28 10:02:10 IST
00003	Cocktail Time	P.G. Wodehouse	1958,06,12	Fiction,Humour,	120	2015-10-28 10:03:54 IST

<< < 1 2 ... 99 100 > >> Go to page: **1** Row count: **10**

Showing 1-10 of 1000

Searching by a primary key

Follow the steps below to search for records of an existing table using primary keys.

1. Log in to the WSO2 DAS management console using the following URL, if you are not already logged in: `https://<DAS_HOST>:<DAS_PORT>carbon/`
2. Click **Main**, and then click **Data Explorer** in the **Interactive Analytics** menu.
3. Select the **Table Name** of which you want to search and view records.
4. Select **By Primary Key** for **Search**.
5. Enter the values of the primary keys defined when you **persisted the event stream**, of which you want to search the records.
6. Click **Search**. You view the filtered records of the selected table as shown below.

Home > Manage > Interactive Analytics > Data Explorer

Help

Data Explorer

Search

Table Name* **BOOK_STORE**

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

Author **Charles Dickens**

Title **A Tale of Two Cities**

Results

Total Records: 0

BOOK_STORE						
ID	Title	Author	Published-Date	Category	Count	timestamp
00001	A Tale of Two Cities	Charles Dickens	['1859','04','01']	Fiction,Classic,	120	2015-10-28 10:00:31 IST

Row count: **10**

Searching by a query

Follow the steps below to search for records of an existing table using an Apache Lucene query.

1. Log in to the WSO2 DAS management console using the following URL, if you are not already logged in. `https://<DAS_HOST>:<DAS_PORT>carbon/`
2. Click **Main**, and then click **Data Explorer** in the **Interactive Analytics** menu.
3. Select the **Table Name** of which you want to search and view records.
4. Select **By Query** for **Search**.
5. Enter the Apache Lucene query which defines the search criteria based on which you want to search the records for **Search Query**.
6. Click **Search**. You view the filtered records of the selected table as shown below.

Search

Table Name* **BOOK_STORE** Schedule Data Purging

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

Title:"The Casual Vacancy"

Search Reset

Results

Total Records: 1 (135 ms)

BOOK_STORE						
ID	Title	Author	Published-Date	Category	Count	_timestamp
00002	The Casual Vacancy	J.K.Rowling	2012-09-27	Fiction,Modern,	120	2015-10-28 10:02:10 IST

<< < 1 > >> Go to page: 1 Row count: 10 Showing 1-1 of 1

Searching by facets

Follow the steps below to search for records of an existing table using facets.

For more information on facets, see [Searching for Data](#).

1. Log in to the WSO2 DAS management console using the following URL, if you are not already logged in: `https://<DAS_HOST>:<DAS_PORT>/carbon`
2. Click **Main**, and then click **Data Explorer** in the **Interactive Analytics** menu.
3. Select the **Table Name** of which you want to search and view records.
4. Select **By Query** for **Search**.
5. Select the values of the facets defined when you persisted the event stream, of which you want to search the records for **Search**.
6. Click **Search**. You view the filtered records of the selected table as shown below.

Search

Table Name* **BOOK_STORE** Schedule Data Purging

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

Search Query

Published-Date **1958 06 12** Select a category Remove

Search Reset

Results

Total Records: 1 (164 ms)

BOOK_STORE						
ID	Title	Author	Published-Date	Category	Count	_timestamp
00003	Cocktail Time	P.G. Wodehouse	1958-06-12	Fiction,Humour,	120	2015-10-28 10:03:54 IST

<< < 1 > >> Go to page: 1 Row count: 10 Showing 1-1 of 1

You can also perform searches with facets using combined Lucene queries in the Data Explorer.

Scheduling data purging

You can schedule data purging tasks using the **Data Explorer**. For instructions on how to schedule data purging, see [Purging Data](#).

Activity Explorer

The activity monitoring dashboard is used to get the list of the events belongs to an activity and search through its results by providing a valid Lucene query.

For example, when a transaction being processed is passing through many subsystems, you can search through

events collected from different subsystems to check whether the transaction is completed, the subsystem at which it is currently being processed etc. The filtering is done by sending events to WSO2 DAS with the same activity ID.

The activity monitoring dashboard groups all the events that belong to the same activity ID and provides you a list of activity IDs. This allows you to search for events by the activity ID and make decisions based on that analysis.

The following sample demonstrates the capabilities of the activity monitoring dashboard.

- Enabling the stream for activity monitoring
- Publish events with the activity ID
- Using the activity monitoring dashboard

Enabling the stream for activity monitoring

In order to use the activity dashboard to search for events, the `activity_id` attribute needs to be included in the relevant event stream, and it should be persisted as a facet as shown below.

Correlation Data Attributes						
Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	correlation.activity_id	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

In the example given above, the **Index Column** check box is also selected for the `activity_id` attribute in order to make it possible to search by this attribute. If there are other attributes by which you want to search, they too need to be persisted as index columns. In this example, it is required to search by the `meta_host`, `meta_http_method`, `meta_message_type`, and `operation_name` attributes, and therefore, the **Index Column** check box is selected for them.

Edit Event Stream

Enter Event Stream Details																																																																												
<input checked="" type="checkbox"/> Persist Event Stream																																																																												
Record Store EVENT_STORE																																																																												
Meta Data Attributes <table border="1"> <thead> <tr> <th>Persist Attribute</th> <th>Attribute Name</th> <th>Attribute Type</th> <th>Primary Key</th> <th>Index Column</th> <th>Score Param</th> <th>Is a Facet</th> </tr> </thead> <tbody> <tr><td><input checked="" type="checkbox"/></td><td>meta_character_set_encoding</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>meta_host</td><td>STRING</td><td><input type="checkbox"/></td><td><input checked="" type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>meta_http_method</td><td>STRING</td><td><input type="checkbox"/></td><td><input checked="" type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>meta_message_type</td><td>STRING</td><td><input type="checkbox"/></td><td><input checked="" type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>meta_remote_address</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>meta_remote_host</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>meta_service_prefix</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>meta_tenant_id</td><td>INTEGER</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>meta_transport_in_url</td><td>INTEGER</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> </tbody> </table>							Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet	<input checked="" type="checkbox"/>	meta_character_set_encoding	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	meta_host	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	meta_http_method	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	meta_message_type	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	meta_remote_address	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	meta_remote_host	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	meta_service_prefix	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	meta_tenant_id	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	meta_transport_in_url	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet																																																																						
<input checked="" type="checkbox"/>	meta_character_set_encoding	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	meta_host	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	meta_http_method	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	meta_message_type	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	meta_remote_address	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	meta_remote_host	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	meta_service_prefix	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	meta_tenant_id	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	meta_transport_in_url	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
Correlation Data Attributes <table border="1"> <thead> <tr> <th>Persist Attribute</th> <th>Attribute Name</th> <th>Attribute Type</th> <th>Primary Key</th> <th>Index Column</th> <th>Score Param</th> <th>Is a Facet</th> </tr> </thead> <tbody> <tr><td><input checked="" type="checkbox"/></td><td>correlation_activity_id</td><td>STRING</td><td><input type="checkbox"/></td><td><input checked="" type="checkbox"/></td><td><input type="checkbox"/></td><td><input checked="" type="checkbox"/></td></tr> </tbody> </table>							Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet	<input checked="" type="checkbox"/>	correlation_activity_id	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>																																																								
Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet																																																																						
<input checked="" type="checkbox"/>	correlation_activity_id	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>																																																																						
Payload Data Attributes <table border="1"> <thead> <tr> <th>Persist Attribute</th> <th>Attribute Name</th> <th>Attribute Type</th> <th>Primary Key</th> <th>Index Column</th> <th>Score Param</th> <th>Is a Facet</th> </tr> </thead> <tbody> <tr><td><input checked="" type="checkbox"/></td><td>SOAPBody</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>SOAPHeader</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>message_direction</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>message_id</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>operation_name</td><td>STRING</td><td><input type="checkbox"/></td><td><input checked="" type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>service_name</td><td>STRING</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr><td><input checked="" type="checkbox"/></td><td>timestamp</td><td>LONG</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> </tbody> </table>							Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet	<input checked="" type="checkbox"/>	SOAPBody	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	SOAPHeader	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	message_direction	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	message_id	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	operation_name	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	service_name	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>														
Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet																																																																						
<input checked="" type="checkbox"/>	SOAPBody	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	SOAPHeader	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	message_direction	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	message_id	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	operation_name	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	service_name	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
<input checked="" type="checkbox"/>	timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																						
Arbitrary Data Attributes <i>No arbitrary data attributes are defined</i>																																																																												
Attribute Name : <input type="text"/>		Attribute Type : STRING	Primary Key : <input type="checkbox"/>	Index Column : <input type="checkbox"/>	Score Param : <input type="checkbox"/>	Is a Facet : <input type="checkbox"/>																																																																						
Advanced																																																																												
Back Save Event Stream																																																																												

For more information, see [Persisting Data for Interactive Analytics](#).

Publish events with the activity ID

As mentioned above, the event stream has the `activity_id` attribute. Each event set to the stream should have a value for this attribute in JSON string. The following is a sample JSON formed activity.

```
[ "1ceccb16-6b89-46f3-bd2f-fd9f7ac447b6" ]
```

Using the activity monitoring dashboard

If you want to perform the search within any time period, select the start time and end time as required in the **From Time** and **To Time** fields. If a time interval is not specified, the search is carried out in all the available records. Therefore, it is recommended to specify a time interval to reduce the system overhead.

Activity Search

Time Search

From Time:

05/25/2015 06:00

are interested on.

To Time:

May 2015

are interested on.

Advanced Search

Search Query:

Su	Mo	Tu	We	Th	Fr	Sa
1	2					
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Time 06:00
Hour
Minute

Activity Search

Time Search

From Time:

05/25/2015 06:00

The time from which activities you are interested on.

To Time:

05/25/2015 13:00

are interested on.

Advanced Search

Search Query:

Su	Mo	Tu	We	Th	Fr	Sa
1	2					
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Time 13:00
Hour
Minute

served.

You also can enter Lucene queries to further filter the results for the time range specified. You can add any number of nested queries that span over multiple tables. A sample query by which you can search is given below.

Activity Search

Time Search

From Time: The time from which activities you are interested on.

To Time: The time to which activities you are interested on.

Advanced Search

Search Query:

```

graph TD
    OR[OR] --> AND[AND]
    AND --> BAM[ORG_WSO2_BAM_ACTIVITY_MONITORING]
    AND --> DAS[ORG_WSO2_DAS_ACTIVITY_MONITORING]
    BAM --> PostBAM[meta_http_method:POST]
    BAM --> HostBAM[meta_remote_host:localhost]
    DAS --> PostDAS[meta_http_method:POST]
    DAS --> HostDAS[meta_remote_host:localhost]
  
```

Buttons:

This fetches a list of activity IDs as shown below. When you click on an activity ID, the first 10 records for that activity are displayed. You can click **More** to view more records for the activity.

Home > Dashboard > Activity Dashboard > activitydashboard ? Help

Activities Search Result

[3ceccb16-6b89-46f3-bd2ffdf9f7ac447b6](#)

ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793621.07870714598000018E-4
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793630.012726495238686785
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793630.4120781945270447
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793630.65822452026018
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793980.013100902903132887
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793550.0013333147387465923
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793630.04424281998493802
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793630.7118232672468965
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793620.06407281607872203
 ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793620.4311797354957233
[more..](#)

[2ceccb16-6b89-46f3-bd2ffdf9f7ac447b6](#)

[5ceccb16-6b89-46f3-bd2ffdf9f7ac447b6](#)

[1ceccb16-6b89-46f3-bd2ffdf9f7ac447b6](#)

[4ceccb16-6b89-46f3-bd2ffdf9f7ac447b6](#)

1 - 5 of 5

Then you can click on each record to view the full record content.

meta_http_method	POST
correlation_activity_id	[3ceccb16-6b89-46f3-bd2ff9f7ac447b6]
operation_name	mediate
meta_remote_address	127.0.0.1
message_id	urn:uuid:c70bae36-b163-4f3e-a341-d7079c58f1ba
meta_transport_in_url	/services/Simple_Stock_Quote_Service_Proxy
service_name	Simple_Stock_Quote_Service_Proxy
meta_remote_host	localhost
meta_service_prefix	https://my:8244
timestamp	1432556278629
message_direction	IN
_version	1.0.0
meta_host	192.168.1.2:9764
SOAPBody	aa
meta_character_set_encoding	UTF-8
meta_tenant_id	123456
SOAPHeader	https://my:8244/services/Simple_Stock_Quote_Service_Proxyurn:uuid:c70bae36-b163-4f3e-a341-d7079c58f1baurn:getFullQuote
meta_message_type	text/xml

[< Back](#)

Query Language Reference

WSO2 DAS allows you to search for persisted events using the [Data Explorer](#). In addition to selecting attributes and categories from lists as shown in [Searching for Data](#), you can write Apache Lucene queries to search for data. This section explains the syntax to be followed when searching for persisted data using Lucene queries.

Query syntax

The following table specifies the query syntax that should be used for different search requirements

Search Requirement	Lucene Query Syntax	Example
View all the data in the selected event table.	Click Search without entering any value in the query field	N/A
Search using a part of the attribute value	Insert the asterisk after the part of the attribute as relevant <code><ATTRIBUTE_NAME>:<PART_OF_ATTRIBUTE_VALUE>*</code>	If you are searching for a book by the title attribute and the actual title is Antony and Cleopatra, you can search by using the following query. <code>title:Antony*</code>
Search using more than one attribute	<code><ATTRIBUTE_NAME>:<ATTRIBUTE_VALUE> AND <ATTRIBUTE_NAME>:<ATTRIBUTE_VALUE></code>	If you are searching for a book written by Ronald Dahl which belongs to the Children's Fantasy category, you can search by the two attributes named author and category using the following query. <code>author:"Ronald Dahl" AND category:"child_fiction"</code>

Search for records that match one of the matching criteria when multiple matching criteria is provided	<ATTRIBUTE_NAME>:<ATTRIBUTE_VALUE> OR <ATTRIBUTE_NAME>:<ATTRIBUTE_VALUE>	If you are searching for a book written by Robin Sharma or a book written by different author on the subject of Leadership, you can search by two attributes named <code>author</code> and <code>subject</code> using the following query. <code>_author:"Robin Sharma" OR subject:"Leadership"</code>
Search for records with a specific attribute value that is within a defined range	<ATTRIBUTE_NAME>:[<MINIMUM_VALUE> TO <MAXIMUM_VALUE>]	If you are searching for a book for which the count is between 100 and 200, you can use the following query. <code>count:[100 TO 200]</code>

This type of search can be carried out only with attributes of which the attribute type is `INT` or `FLOAT`.

For detailed information about the Lucene syntax, see [Apache Lucene - Query Parser Syntax](#).

It is not possible to search for attributes defined as facets directly using Lucene queries, but should be separately given in the Data Explorer, or used with the [Analytics REST API](#).

WSO2 DAS Lucene Query Extensions

Timestamp Operations on Fields

WSO2 DAS supports only the primitive data types when persisting data. Therefore it does not have a special data type for timestamp. The `LONG` data type should be used when sending timestamp values. However, because it is convenient to query with a string time stamp format instead of querying for a long value which is exactly the same as the actual value, the following string format is supported.

`YYYY-MM-dd HH:mm:ss z`

e.g., `2015-01-02 15:22:10 GMT+6`

This value is converted to a Unix timestamp long value and then used for searching.

Sample Queries

- The following query can be used to search for a person whose last name is Smith and the date of birth is 2nd of January 2015.

```
surname:"Smith" AND birthdate:"2015-01-02"
```

- The following query can be used for a log level WARN which occurred at a time between the time stamps 2015-10-01 01:05:20 and 20.15-12-15 00:00:00.
`log_level: "WARN" AND timestamp: [2015-10-01 01:05:20 TO 2015-12-15 00:00:00]`

Search with Multi-Word Values

If an attribute of the STRING type is instructed to be indexed, Apache Lucene tokenises and indexes it in a manner that allows you to search by that attribute using individual word values. If you want to search by the attribute by using the complete attribute value, WSO2 DAS allows the following query to be used for this purpose.

```
_X:<EXACT_ATTRIBUTE_Value>
```

In this query, X = Attribute Name

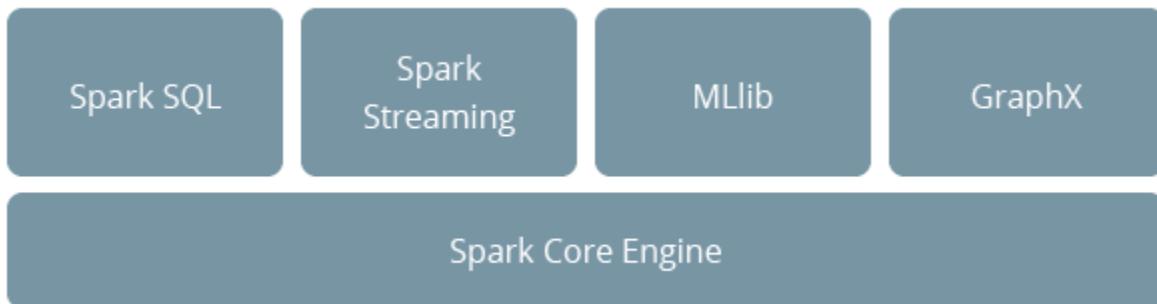
Sample query

If the `Name` attribute is indexed and a record with `Will Smith` as the value of the `Name` attribute is stored:

- Use the following query to search for all records with `Smith` as all or part of the value for the `Name` attribute.
`name: "Smith"`
- Use the following query to search for all records where `Will Smith` is the exact value for the `Name` attribute
`_name: "Will Smith"`

Batch Analytics Using Spark SQL

Apache Spark is a powerful open source processing engine built around speed, ease of use, and sophisticated analytics. WSO2 DAS employs [Apache Spark](#) as its analytics engine. Further, WSO2 DAS 3.0.0 extends the latest Spark API (version 1.2.1) to come up with its data analytics processor replacing [Apache Hadoop](#). The ecosystem of Apache Spark is as follows.



For more information on Apache Spark, see [Apache Spark documentation](#).

Following sections describe how you can perform batch analytics using Apache Spark SQL in WSO2 DAS.

- Batch Analytics Console
- Scheduling Batch Analytics Scripts
- Spark Query Language
- Publishing Events Using Apache Spark
- Creating Spark User Defined Functions
- Creating Spark User Defined Aggregate Functions
- Spark Troubleshooting
- Incremental Processing

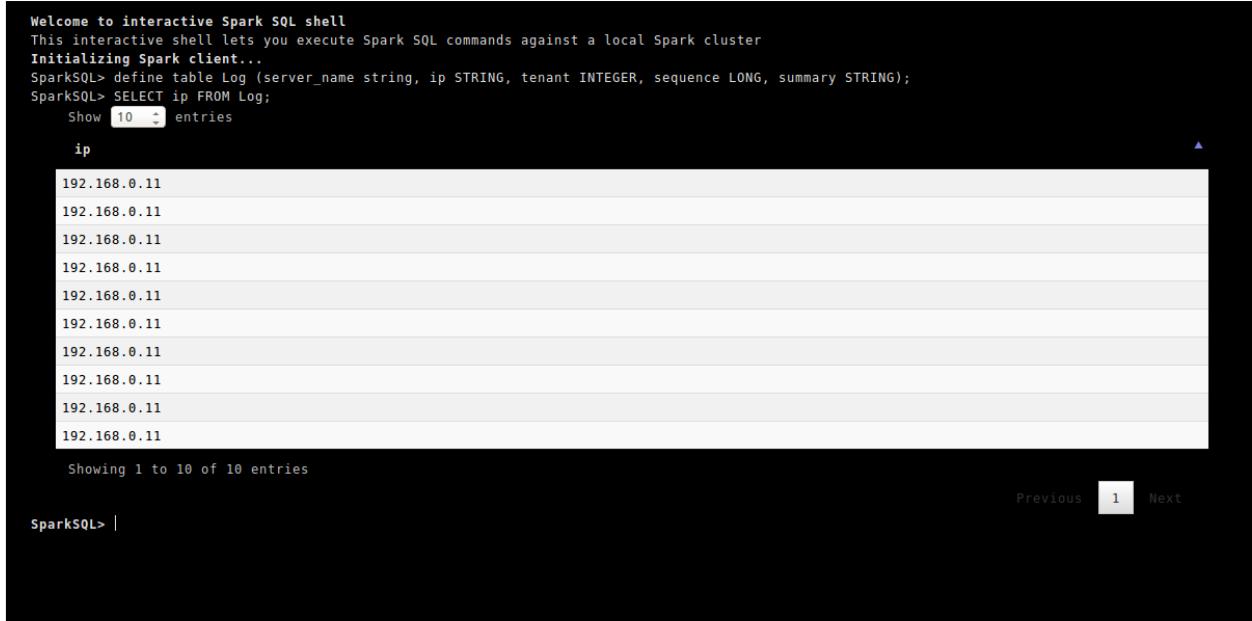
- Monitoring Spark Performance

Batch Analytics Console

Batch Analytics Console is an interactive tool where the users can enter any [supported Spark queries](#) to the console. Spark processes the entered query internally, and then displays the results in the screen immediately. Follow the steps below to use this tool.

1. Log in to the WSO2 DAS management console.
2. In the **Main** tab, click **Console**.
3. In the Spark Console view, enter the Spark query and execute it as shown below.

Interactive Spark Console



```
Welcome to interactive Spark SQL shell
This interactive shell lets you execute Spark SQL commands against a local Spark cluster
Initializing Spark client...
SparkSQL> define table Log (server_name string, ip STRING, tenant INTEGER, sequence LONG, summary STRING);
SparkSQL> SELECT ip FROM Log;
Show 10 entries
ip
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
Showing 1 to 10 of 10 entries
Previous 1 Next
SparkSQL> |
```

Scheduling Batch Analytics Scripts

Apache Spark is used as the core analytics engine in DAS 3.0.0. For information on writing Spark queries to analyze the collected data, see [Data analysis using SQL](#).

Analytics scripts

Analytics scripts are used when you have to execute a set of Spark queries in a sequence. Also, you can schedule a Analytics script, to trigger it to execute the query automatically in a given period of time. (E.g. fire at 12 (noon) every day, or fire at every minute starting at 2 p.m. and ending at 2:59 p.m. every day etc.). You need to configure this scheduled time using a cron expression. For more information about cron expressions, see [Cron Trigger Tutorial](#).

You can add/edit/delete scripts, and also you can provide your schedule time for your script to execute as described below.

Adding a new script

Follow the steps below to add a new Spark script.

1. Log in to the WSO2 DAS Management Console.
2. In the **Main** tab, click **Scripts** to open the **Available Analytics Scripts** page.
3. Click **Add New Analytics Script** to open the **Add New Analytics Script** page. Then enter the following details related to your script as shown in the example below.

Add New Analytics Script

New Analytics Script Information

Script Name *	<input type="text" value="MyFirstAnalyticsScript"/> <small>The name of the actual analytics script.</small>
Spark SQL Queries *	<pre> 1 define table Log (server_name string, ip STRING, tenant INTEGER, sequence LONG, summary STRING); 2 SELECT ip FROM Log; 3 SELECT server_name, count(*) FROM Log GROUP BY server_name; 4 SELECT COUNT(*) FROM Log WHERE summary LIKE '%Joe%'; 5 SELECT substr(summary, 1, 5) FROM Log; 6 SELECT LAST(ip) FROM Log; </pre> <small>Position: Ln 6, Ch 26 Total: Ln 6, Ch 269</small> <input checked="" type="checkbox"/> Toggle editor
<p>Schedule Script</p> <p>Cron Expression <input type="text" value="0 0/5 * * * ?"/> <small>The cron expression for the rate of analytics script to be executed</small></p>	

Script Name	MyFirstAnalyticsScript
Spark SQL Queries	<pre> define table Log (server_name string, ip STRING, tenant INTEGER, sequence LONG, summary STRING); SELECT ip FROM Log; SELECT server_name, count(*) FROM Log GROUP BY server_name; SELECT COUNT(*) FROM Log WHERE summary LIKE '%Joe%'; SELECT substr(summary, 1, 5) FROM Log; SELECT LAST(ip) FROM Log; </pre>
Cron Expression	<p>0 * * * * ?</p> <p>This cron expression defines the schedule time of the script to execute it in every minute. From the time you save the script, the script will be executed at the beginning of every minute. (E.g.:10:21:00, 10:22:00, 10:23:00,..)</p>

4. Click **Execute**, to execute the provided queries. This will display the results as follows.

The screenshot shows a code editor window with the following SQL queries:

```

9 | SELECT substr(summary, 1, 5) FROM Log;
10| 
11| SELECT LAST(ip) FROM Log;
12| 
13|

```

Below the editor, the status bar shows "Position: Ln 13, Ch 5" and "Total: Ln 13, Ch 296". There is also a "Toggle editor" checkbox.

Below the editor is a "Schedule Script" section with a "Cron Expression" field containing "0 * * * ?" and a note: "Enter the cron expression for the rate of spark script to be executed".

At the bottom of this section are three buttons: "Add", "Execute Script" (which is highlighted with a red box), and "Cancel".

Below this is an "Execution Results" section. It shows the query "define table Log (server_name string, ip STRING, tenant INTEGER, sequence LONG, summary STRING)" followed by a "Query Executed" message with an information icon.

Then it shows the query "SELECT ip FROM Log" and the results table:

ip
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11

- Click **Add**, to add the configured script.

When a Spark script is created and saved, it is stored in the registry. It can be accessed using the `/_system/config/repository/components/org.wso2.carbon.analytics.spark` registry path. For more information about the registry, see [Registry](#).

Editing a script

Follow the steps below to edit an existing Analytics script.

- Log in to the WSO2 DAS Management Console.
- In the **Main** tab, click **Scripts** to open the **Available Analytics Scripts** page.
- Click **Edit** for the script you want to edit.

The screenshot shows the "Available Analytics Scripts" page with two entries:

Scripts	Actions
MyFirstAnalyticsScript	Edit Execute Execute in Background Delete
SparkScript	Edit Execute Execute in Background Delete

Below the table is a green "Add New Analytics Script" button.

- Change the content of the script as required. You can update the scheduling information as well.

When you do not enter any value for the scheduling time, then your script is not scheduled to execute. However, if you want to ensure that your script is valid, click **Execute**. This will execute the queries that you give in the queries window.

For example, you can edit the script created above to unschedule the scheduled time as shown below.

Analytics Script Information

Spark SQL Queries *

```

1 define table Log (server_name string, ip STRING, tenant INTEGER, sequence LONG, summary STRING);
2 SELECT ip FROM Log;
3
4
5
6
7
8

```

Position: Ln 3, Ch 1 | Total: Ln 4, Ch 143 | Toggle editor

Schedule Script

Cron Expression

0 0/5 * * * ? The cron expression for the rate of analytics script to be executed

Update Execute Script Execute in Background Cancel

- Click **Update** to save the changes as shown above.

Deleting a script

Follow the steps below to delete an Analytics script.

- Log in to the WSO2 DAS Management Console.
- In the **Main** tab, click **Scripts** to open the **Available Analytics Scripts** page.
- Click **Delete** for the script you want to delete.

Home > Manage > Batch Analytics > Scripts Help

Available Analytics Scripts

Scripts	Actions
MyFirstAnalyticsScript	Edit Execute Execute in Background Delete
SparkScript	Edit Execute Execute in Background Delete

Add New Analytics Script

- Click **Yes** in the dialog box which appears to confirm deletion.

If you delete the script you cannot undo that operation, and it will be completely removed from the system. Also, deleting the script will delete the scheduled task associated with it.

Executing a script

You can execute the script manually when you are adding/editing the script, without using any scheduled task. This will trigger the execution of the script content provided in the queries window at that moment. Also, you can execute the script content out of the edit mode as shown below. During this operation, WSO2 DAS fetches the script content and gives it to Spark to execute all the queries in the script. Once the execution is completed the results are displayed.

Follow the steps below to execute the script content.

- Log in to the WSO2 DAS Management Console.
- In the **Main** tab, click **Scripts** to open the **Available Analytics Scripts** page.
- Click **Execute** for the script you want to execute.

Scripts	Actions
MyFirstAnalyticsScript	Edit Execute Execute in Background Delete
SparkScript	Edit Execute Execute in Background Delete

[+ Add New Analytics Script](#)

4. Now, the script execution job is immediately dispatched to Spark engine. It will display the results once the job is completed as shown below.

Script Executor – MyFirstAnalyticsScript

Analytics Script Results

```
Query: define table Log (server_name string, ip STRING, tenant INTEGER, sequence LONG, summary STRING)
Query Executed
```

Query: SELECT ip FROM Log

Results:

ip
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11
192.168.0.11

Spark Query Language

Interactive SQL (Structured Query Language) queries are widely used for exploring and analyzing data in the current context by many business intelligence users. WSO2 DAS 3.0.0. ships with the feature of running SQL queries on the underlying datasources as specified in the [DAS Data Access Layer \(DAL\)](#).

It uses [Spark SQL](#) as the query engine, which succeeds Apache Hive from WSO2 DAS 3.0.0 onwards. This provides a powerful integration with the rest of the Spark analytics engine. For more information on Spark SQL, see [Spark SQL Programming Guide](#).

- [Spark SQL queries](#)
- [WSO2 DAS SQL guide](#)
- [Reserved words in Spark SQL](#)
- [Reserved words in the WSO2 Carbon environment](#)

Spark SQL queries

Spark SQL follows the standard SQL format. For information on the syntax explanations of the standard SQL format, see [SQL Syntax](#).

Some query types of the standard SQL format are not supported by Spark SQL.

The query types that are supported by the Spark SQL parser are yet to appear in the published docs by the Apache Spark project. For more information on the SparkSQL query syntax, see the [SparkSQL parser code](#), and the [SQL Query test suite](#).

WSO2 DAS SQL guide

WSO2 DAS inherits the query parsing options from the Spark SQL's native query parser. Click on the relevant tab to view the query formats to be used for the required action.

[Create table queries](#)[Insert queries](#)[Select queries](#)[Incremental Processing Queries](#)

Use the following query syntax to register a temporary table in the Spark environment using data from Carbon analytics or any other relation provider class.

```
CREATE TEMPORARY TABLE <table_name>
USING <provider_name>
OPTIONS ( <options> )
AS <alias>;
```

The parameters of the above syntax are described below.

Element	Description
<table_name>	Name of the table which is created in the Spark environment.
<provider_name>	Provider of data to create the table. It can be either Carbon analytics or a relation provider class.
<options>	Other options for Spark to refer when creating the table.
<alias>	An alias to uniquely identify the created table. This is optional.

The provider used to create the temporary table can be Carbon Analytics, Carbon JDBC, Compressed Event Analytics or other. See the following sections for detailed information on how to use each provider to create the table.

- [Creating the Table Using Carbon Analytics as the Provider](#)
- [Creating the Table Using Carbon JDBC as the Provider](#)
- [Creating the Table Using Compressed Event Analytics as the Provider](#)
- [Creating the Table Using Other Relation Providers](#)

Use the following query syntax to insert data into the temporary tables that already exist in the Spark environment.

```
INSERT INTO/OVERWRITE TABLE <table_name> <SELECT_query>
```

Parameters of the above syntax are described below.

Parameter	Description
<table_name>	The name of the temporary table you want to insert values into.
<SELECT_query>	The select statement used to enter values into the temporary table being overwritten.

For example;

```
INSERT OVERWRITE TABLE plugUsage
select house_id, household_id, plug_id, max(value) - min (value) as usage,
compositeID(house_id, household_id, plug_id) as composite_id from debsData where
property = false group by house_id, household_id, plug_id;
```

You can use any SELECT query in the standard SQL syntax to select data from a table which is created in the Spark environment.

```
SELECT * from <temp_table>;
```

<temp_table> parameter specifies the name of the temporary table from which data should be selected.

From DAS 3.1.0 onwards, it is possible to process data incrementally based on the timestamp provided. Incremental processing is only available for the Carbon Analytics relation provider. It should be enabled per table via the `IncrementalParams` option. For more information about this option, see [Creating the Table Using Carbon Analytics as the Provider](#).

The incremental processing queries that can be carried out are as follows.

Query	INCREMENTAL_TABLE_COMMIT
Description	This query sets the incremental metadata. It should be set at a time when processing a set of data is complete. Once this query is committed, only the new data from the last processed data row is processed.
Syntax	INCREMENTAL_TABLE_COMMIT <incremental ID1, incremental ID2, ...>
Example	incremental_table_commit TableID1, TableID2;

Query	INCREMENTAL_TABLE_SHOW
Description	This query shows the incremental metadata for the given tables.
Syntax	INCREMENTAL_TABLE_SHOW <incremental ID1, incremental ID2, ...>
Example	incremental_table_show TableID1, TableID2;

Query	INCREMENTAL_TABLE_RESET
Description	This query resets the incremental metadata for the given tables. It resets the incremental metadata values to Long.MIN_VALUE.
Syntax	INCREMENTAL_TABLE_RESET <incremental ID1, incremental ID2, ...>
Example	incremental_table_reset TableID1, TableID2;

Reserved words in Spark SQL

The following are the reserved words in Spark SQL by default. These words cannot be used in Data Definition Language (DDL) tasks (e.g., as column names, etc).

The reserved words are case insensitive

- ABS
- ALL
- AND
- APPROXIMATE
- AS
- ASC
- AVG
- BETWEEN
- BY
- CASE
- CAST
- COALESCE
- COUNT
- DESC
- DISTINCT
- ELSE
- END
- EXCEPT
- FALSE
- FIRST
- FROM
- FULL
- GROUP
- HAVING
- IF
- IN
- INNER
- INSERT
- INTERSECT
- INTO
- IS
- JOIN
- LAST
- LEFT
- LIKE
- LIMIT
- LOWER
- MAX
- MIN
- NOT
- NULL
- ON
- OR
- ORDER
- SORT
- OUTER
- OVERWRITE
- REGEXP
- RIGHT

- RLIKE
- SELECT
- SEMI
- SQRT
- SUBSTR
- SUBSTRING
- SUM
- TABLE
- THEN
- TRUE
- UNION
- UPPER
- WHEN
- WHERE
- WITH

Reserved words in the WSO2 Carbon environment

The following words are reserved in the WSO2 Carbon environment.

The reserved words are case sensitive

- CarbonAnalytics
- CarbonJDBC

Creating the Table Using Carbon Analytics as the Provider

Use the following query to create a table in the Spark environment (if it does not already exist), using data from Carbon analytics. Carbon analytics refer to either the built-in H2 database or any external database that is connected to the DAL.

```
CREATE TEMPORARY TABLE plugUsage
USING CarbonAnalytics
OPTIONS (tableName "plug_usage",
         schema "house_id INT, household_id INT, plug_id INT, usage FLOAT -sp,
composite FACET -i",
         primaryKeys "household_id, plug_id"
        );
```

Carbon analytics relation provider options

The options that can be used with the Carbon analytics relation provider are described below.

Specify the options in key value pairs separated by commas, and give the values within quotation marks.

Option	Description	Example
tableName or streamName	Name of the table in the DAL.	tableName "plug_usage" or streamName "plug.usage"

schema	<p>Schema of the table in the DAL. This is mandatory.</p> <div style="border: 1px solid #ccc; padding: 10px;"> <p>You must specify the schema for the table. This was not required in DAS 3.0.1 but required in DAS 3.1.0 upwards.</p> </div> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>Schema fields are column name and column type value pairs with indexing options. These fields should be comma separated. Following are the schema indexing options.</p> <ul style="list-style-type: none"> • <code>-i</code> denotes an indexed column. • <code>-sp</code> denotes an indexed column with score param. This column should be of numeric type. </div> <p>The following fields are special fields in an analyticstable:-</p> <ul style="list-style-type: none"> • <code>_timestamp</code> denotes the timestamp of the record when it is persisted • <code>_tenantId</code> denotes the tenant id of the record 	<pre>schema "house_id INT, household_id INT, plug_id INT, usage FLOAT -sp, composite FACET -i"</pre>
primaryKeys	Primary key of the table in the DAL. This is optional. Assign primary keys if and only if you have provided a schema.	<pre>primaryKeys "household_id, plug_id"</pre>
mergeSchema	A boolean flag used for schema merging. If this option is set to <code>true</code> , the given schema is merged with the corresponding table schema in the Data Access Layer (if a schema exists). If the option is set to <code>false</code> , the given schema overwrites the table schema in the Data Access Layer.	<pre>mergeSchema "false"</pre>
recordStore	<p>The Analytics Record Store in which this table is created.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The default Analytics Record Store used by CarbonAnalytics is the <code>PROCESSED_DATA_STORE</code>.</p> </div>	<pre>recordStore "EVENT_STORE"</pre>
globalTenantAccess	A boolean value which represents a flag which says, if "true", read data records from all the tenants with the given table name, and also write to the same table, where by looking at the incoming records' <code>_tenantId</code> value, it will route the records to the specified tenant's table. A user can use this flag, and filter/groupby tenants from Spark queries. When creating a table with this option enabled, in the schema, the field <code>_tenantId</code> should be added with the type <code>INTEGER</code> , in order to read/write tenant data.	<pre>globalTenantAccess "true"</pre>

incrementalParams	<p>This is a set of parameters that govern the incremental data processing (time based). The parameters are as follows.</p> <ul style="list-style-type: none"> Incremental Table ID: This is an alias ID for the table to store incremental data. Incremental Table Unit: This parameter sets the start timestamp of the data to be processed. Possible values are SECOND, MINUTE, HOUR, DAY, MONTH and YEAR. Incremental Table Unit Offset: This parameter specifies the number of incremental table units to be processed from the last processed event. The default value is 1. The value should always be greater than 0. 	<ul style="list-style-type: none"> incrementalParams "pUsage ,MINUTE" e.g., If the last processed row has the timestamp 15:10:21 .555 of 1/1/2016, the processing starts from 15:10:00.000 of 1/1/2016. incrementalParams "pUsage ,MINUTE, 5" e.g., If the last processed row has the timestamp 15:10:21 .555 of 1/1/2016, then the processing starts from 15:05:00 .000 of 1/1/2016.
-------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Creating the Table Using Carbon JDBC as the Provider

The CarbonJDBC provider can be used to interact directly with a relational database, bypassing the DAS Data Access Layer. As a result, the data is stored in the format native to the relational storage mechanism in use without compression or encoding. This is useful in scenarios where the results of a Spark query need to be stored in a relational database to be accessed by third parties such as reporting tools or dashboards, or if the results need to be readable outside WSO2 DAS.

Multi-tenancy with the CarbonJDBC provider

To use the CarbonJDBC analytics provider for tenant scenarios, you will need to specifically create the relevant data source that you intend to use for that tenant. Please see the [page on data sources](#) for instructions on creating a data source.

The following syntax is used to create a temporary table using the CarbonJDBC analytics provider in Apache Spark with a relational table backing it.

```
CREATE TEMPORARY TABLE <temp_table> using CarbonJDBC options (dataSource "<datasource name>", tableName "<table name>", schema "<schema>" [, primaryKeys "<primaryKeys>"] );
```

Options in the above syntax are described below.

Option	Condition	Description
dataSource "<datasource name>"	Mandatory	<p>The name of the data source from which data should be obtained for the table.</p> <p>Only RDBMS data sources are supported.</p>

tableName " <table name> "	Mandatory	<p>The name of the table in the selected data source which should be used to temporary table. This would be the table from which data will be obtained on SELECT operations, and to which data will be added on INSERT operations.</p> <p>If the table does not already exist in the selected data source, it will be automatically based on the given schema together with the relevant type map of the RDBMS instance in use.</p>
schema " <schema> "	Mandatory	<p>The schema used to represent the relational table within Apache Spark.</p> <p>The data types allowed for this parameter are as follows.</p> <ul style="list-style-type: none"> • BINARY • BOOLEAN • BYTE • DATE • DOUBLE • FLOAT • INTEGER • LONG • NULL • SHORT • STRING • TIMESTAMP <p>The STRING data type also supports the optional specification of field size which overrides the default value set in the <DAS_HOME>/repository/conf/analytics/spark/spark-jdbc-config.xml file.</p> <p>The specification of the -i key following a particular column definition defines a primary index on the target datasource. If multiple parameters are specified, all of them are grouped to create a single index.</p>
primaryKeys " <primaryKeys> "	Optional	This parameter can be used to specify unique keys that are either in use or required by the table.

The field mappings for each supported database type are maintained in the <DAS_HOME>/repository/conf/analytics/spark/spark-jdbc-config.xml file.

Sample Queries

The following are some sample queries using CarbonJDBC as the provider.

```

CREATE TEMPORARY TABLE StateUsage using CarbonJDBC OPTIONS (dataSource
"MY_DATASOURCE", tableName "state_usage", schema "us_state STRING -i, polarity INTEGER,
usage_avg FLOAT", primaryKeys "us_state");

INSERT INTO TABLE StateUsage SELECT state, polarity, state_avg_usage FROM
USCensusData;

INSERT OVERWRITE TABLE StateUsage SELECT state, polarity, state_avg_usage FROM
USCensusData;

SELECT * FROM StateUsage LIMIT 5;

```

Creating the Table Using Compressed Event Analytics as the Provider

The main purpose of the Compressed Event Analytics Relational Provider is to split the aggregated events published to DAS into single events and load them to Apache Spark. In order to use the Compressed Event Analytics Relational Provider, the Data Access Layer (DAL) table should have the following two columns.

- A meta field named `meta_compressed` indicating whether the data in the other column is compressed or not.
- The `flowData` column that contains the aggregated events. Data in this column may be compressed (i.e. zipped, and followed by the Base64 encoding).

The aggregated data in the `flowData` column should be structured as shown below.

```
{
  'events': [
    {
      'componentType': 'Proxy Service',
      'componentId': 'TestProxy',
      'children': [1],
      'entryPoint': 'TestProxy'
    },
    {
      'componentType': 'Sequence',
      'componentId': 'PROXY_INSEQ',
      'children': [2],
      'entryPoint': 'TestProxy'
    }
  ],
  'messageFlowId': 123456,
  'payloads': [
    {
      'payload': '?',
      'events': [
        {
          'eventIndex': 16,
          'attribute': 'beforePayload'
        }
      ]
    },
    {
      'payload': '?',
      'events': [
        {
          'eventIndex': 15,
          'attribute': 'beforePayload'
        }
      ]
    }
  ]
}
```

Use the following query syntax to create a table in the Spark environment (i.e. if it does not already exist) using data from Compressed Events Analytics.

```
CREATE TEMPORARY TABLE <temp_table_name>
USING CompressedEventAnalytics
OPTIONS (tableName <table_name>, schema "componentType STRING, componentId STRING,
children STRING, entryPoint STRING -sp, messageFlowId FACET
```

Compressed Event Analytics relation provider options

The options that can be used with the Compressed Event Analytics Relation Provider are described in the table

below.

Specify the options in key value pairs separated by commas, and enter the values within quotation marks.

Option	Description	Example
tableName/streamName	The name of the table in the DAL.	tableName "plug_usage" or streamName "plug.usage"
schema	<p>The schema of the temporary table. It is required to specify this.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> Schema names are column name and column type value pairs with indexing options. These fields should be comma separated. </div> <p>Schema indexing options are as follows.</p> <ul style="list-style-type: none"> • -i denotes an indexed column. All indexed columns should be of the numeric type. • -sp denotes an indexed column with a score parameter. <p>The following fields are special fields in an analytics table:-</p> <ul style="list-style-type: none"> • _timestamp denotes the timestamp of the record when it is persisted • _tenantId denotes the tenant id of the record 	schema "componentType STRING, componentId STRING, children STRING, entryPoint STRING -sp, messageFlowId FACET -i"
mergeSchema	This is a boolean flag used for schema merging. If this option is set to true, the given schema is merged with the corresponding table schema in the DAL (if a schema exists). The default value is "false".	mergeSchema "false"

Creating the Table Using Other Relation Providers

Use the following query syntax to create a table in the Spark environment, using data from a relation provider class. A relation provider builds the connection from Spark to any external database. For example, the following query creates a table in the Spark environment using the Spark JDBC provider connecting to a H2 database.

```
CREATE TEMPORARY TABLE foo
USING jdbc
OPTIONS (url "jdbc:h2:mem:testdb0",
          dbtable "TEST.PEOPLE",
          user "testUser",
          password "testPass"
        );
```

Other relation provider options

For more information on the options that can be used with the Spark JDBC relation provider, see [Spark SQL and DataFrame Guide](#).

Specify the options in key value pairs separated by commas, and give the values within quotation marks.

Publishing Events Using Apache Spark

You can publish events from WSO2 DAS using Spark SQL queries. This feature alerts any interested party whenever the content of an existing Spark table is changed. e.g., A scheduled Spark script can periodically check whether a Spark table meets specific conditions. If the conditions are satisfied, an event can be published downstream to notify the interested parties. The following are the functions carried out to publish events from WSO2 DAS using Apache Spark.

- [Creating the event stream to publish events from Spark](#)
- [Creating the event receiver](#)
- [Publishing events to the event stream](#)

Creating the event stream to publish events from Spark

To publish events from Spark, an event stream with the required stream attributes (i.e., attributes for which values are published from Spark) should be defined as the first step. A sample event stream definition is as follows. For more information on event streams, see [Understanding Event Streams and Event Tables](#).

```
{
  "streamId": "TestEventStream:1.0.0",
  "name": "TestEventStream",
  "version": "1.0.0",
  "nickName": "TestStream",
  "description": "Test Stream",
  "metaData": [],
  "correlationData": [],
  "payloadData": [
    {
      "name": "ip",
      "type": "STRING"
    },
    {
      "name": "name",
      "type": "STRING"
    },
    {
      "name": "testMessage",
      "type": "STRING"
    }
  ]
}
```

Creating the event receiver

Once you define the event stream, you need to create an event receiver of the `WSO2Event` type to receive events from Spark. For more information, see [WSO2Event Event Receiver](#).

Publishing events to the event stream

Use the following Spark SQL queries to create a virtual table in the Spark table space to hold the published events, and to publish the rows of it into the defined event stream as events. The `org.wso2.carbon.analytics.spark.core.util.EventStreamProvider` class works as the bridge between the existing Spark table and DAS event stream storage to fetch data from the existing Spark table and publish events to the defined event stream.

```

CREATE TEMPORARY TABLE <table_name>
USING org.wso2.carbon.analytics.spark.event.EventStreamProvider
OPTIONS (streamName "<stream_name>",
         version "<stream_version>",
         payload "<payload>")
;
INSERT OVERWRITE TABLE <table_name> <select_query>;

```

The parameters of the above query are described below.

Parameter	Description
<table_name>	The name of the Spark table that is mapped with the created event stream.
<stream_name>	The name of the stream that should send events.
<stream_version>	The version number of the stream that sends events.
<payload>	A string containing stream attributes as comma-separated pairs with the name of the attribute and its data type in the following format: [<attribute_name><space><attribute_type>] (e.g., payload "ip STRING, name STRING, testMessage STRING ")
<select query>	The select query to filter the required fields that should be published to the event stream from the existing Spark table. (e.g., select ip_address, name, message from EventStream)

Once these functions are carried out, you can attach an event publisher (such as `email` or `JMS`) to the published events stream and get the events delivered to a preferred location. For detailed instructions to configure publishers, see [Creating Alerts](#).

Creating Spark User Defined Functions

Apache Spark allows UDFs (User Defined Functions) to be created if you want want to use a feature that is not available for Spark by default. WSO2 DAS has an abstraction layer for generic Spark UDF (User Defined Functions) which makes it convenient to introduce UDFs to the server.

The following query is an example of a custom UDF.

```
SELECT id, concat(firstName, lastName) as fullName, department FROM employees;
```

The steps to create a custom UDF are as follows.

- Step 1: Create the POJO class
- Step 2: Package the class in a jar
- Step 3: Update Spark UDF configuration file

Step 1: Create the POJO class

The following example shows the UDF POJO for the StringConcatonator custom UDF class. The name of the Spark UDF should be the name of the method defined (concat in this example). This will be used when calling the UDF with Spark. e.g., `concat("custom", "UDF")` returns the String "Custom UDF".

```

/**
 * This is an UDF class supporting string concatenation for spark SQL
 */
public class StringConcatonator {

    /**
     * This UDF returns the concatenation of two strings
     */
    public String concat(String firstString, String secondString) {
        return firstString + secondString;
    }
}

```

- Apache Spark does not support primitive data type returns. Therefore, all the methods in a POJO class should return the wrapper class of the corresponding primitive data type.

e.g., A method to add two integers should be defined as shown below.

```

public Integer AddNumbers(Integer a)
{
}

```

- Method overloading for UDFs is not supported. Different UDFs should have different method names for the expected behaviour.
- If the user consumes a data type that is not supported for Apache Spark, the following error appears when you start the DAS server.

```
Error initializing analytics executor: Cannot determine the return
DataType
```

For a list of return types supported for Apache Spark, see [Spark SQL and DataFrames and Datasets Guide - Data Types](#).

If you need to use one or more methods that are not UDF methods, and they contain return types that are not supported for Apache Spark, you can use a separate class to define them. This class does not have to be added to the <DAS_HOME>/repository/conf/analytics/spark/spark-udf-config.xml file.

Step 2: Package the class in a jar

The custom UDF class you created should be bundled as a jar and added to <DAS_HOME>/repository/components/lib directory.

Step 3: Update Spark UDF configuration file

Add the newly created custom UDF to the <DAS_HOME>/repository/conf/analytics/spark/spark-udf-config.xml file as shown in the example below.

```

<udf-configuration>
  <custom-udf-classes>
    <class-name>org.james.customUDFs.StringConcatonator</class-name>

    <class-name>org.wso2.carbon.analytics.spark.core.udf.defaults.TimestampUDF</class-name>
  >
  </custom-udf-classes>
</udf-configuration>

```

This configuration is required for Spark to identify and use the newly defined custom UDF.

Spark adds all the methods in the specified UDF class as custom UDFs.

Creating Spark User Defined Aggregate Functions

Apache Spark UDAFs (User Defined Aggregate Functions) allow you to implement customized aggregate operations on Spark rows. Custom UDAFs can be written and added to DAS if the required functionality does not already exist in Spark.

In addition to the definition of custom Spark UDAFs, WSO2 DAS also provides an abstraction layer for generic Spark UDAF functionality that be used to introduce custom UDAFs to the DAS/Spark runtime without changes to the server configuration.

The following query is an example of a custom UDAF named `geometricMean`.

```
SELECT geometricMean(price) as meanPrice FROM products;
```

- Apache Spark 1.6.2, and by extension WSO2 DAS 3.1.0, do not support primitive data type returns. Therefore, all the methods in a POJO class should return the wrapper class of the corresponding primitive data type.
- Apache Spark 1.6.2 also does not differentiate between algebraic and non-algebraic UDAFs. Therefore, all the methods in the parent class need to be implemented for all the UDAFs. e.g., Mathematical operations such as median do not support merge operations, but they are still required.
- Method overloading for UDAFs is not supported. Different UDAFs should have different method names for the expected behaviour. e.g., a UDAF with the same functionality for `String` and `Number` types need to be defined as two separate UDAF implementations, where each handles a different type.
- For a list of return types supported for Apache Spark, see [Spark SQL and DataFrames and Datasets Guide - Data Types](#).

There are two patterns that can be followed to create a custom UDAF. Click on the relevant tab based on the pattern you want to follow.

[Manual UDAF Installation](#) [Automatic UDAF Installation](#)

This pattern involves creating and packaging the UDAF artifact, deploying it to the WSO2 DAS environment and then updating the Spark UDF configuration file so that WSO2 DAS recognizes the deployed UDAF at runtime.

Follow the steps below to create a custom UDAF.

Step 1: Create the POJO class

The following is an example of a UDAF POJO for the `geometricMean` custom UDAF class.

```

package com.whitesnake.analytics.udaf;

import org.apache.spark.sql.Row
import org.apache.spark.sql.expressions.{MutableAggregationBuffer,
UserDefinedAggregateFunction}
import org.apache.spark.sql.types._

/**
 * This UDAF could be used to calculate the geometric mean of the given set of
numbers.
 */
class GeometricMeanUDAF extends UserDefinedAggregateFunction {

    override def inputSchema: org.apache.spark.sql.types.StructType =
StructType(StructField("value", DoubleType) :: Nil)

    override def bufferSchema: StructType = StructType(
StructField("count", LongType) :: StructField("product", DoubleType) :: Nil
)

    override def dataType: DataType = DoubleType

    override def deterministic: Boolean = true

    override def initialize(buffer: MutableAggregationBuffer): Unit = {
buffer(0) = 0L
buffer(1) = 1.0
}

    override def update(buffer: MutableAggregationBuffer, input: Row): Unit = {
buffer(0) = buffer.getAs[Long](0) + 1
buffer(1) = buffer.getAs[Double](1) * input.getAs[Double](0)
}

    override def merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit = {
buffer1(0) = buffer1.getAs[Long](0) + buffer2.getAs[Long](0)
buffer1(1) = buffer1.getAs[Double](1) * buffer2.getAs[Double](1)
}

    override def evaluate(buffer: Row): Any = {
math.pow(buffer.getDouble(1), 1.toDouble / buffer.getLong(0))
}
}

```

This class implements the functionality of the Spark `org.apache.spark.sql.expressions.UserDefinedAggregateFunction` abstract class, and therefore, the functionality of the following should be implemented.

- `inputSchema`: The schema of the input rows.
- `bufferSchema`: The schema of intermediate results.
- `dataType`: The datatype of the final result.
- `Deterministic`: This denotes whether the same inputs always produce the same results.

- `initialize()`: This is called once per node for a given group.
- `update()`: This is called once per input record.
- `merge()`: This is called to compute partial results and combine them together.
- `evaluate()`: This is called to compute the final result.

For more information on each of the above parameters, see the [official Spark Javadoc on UDAFs](#).

Step 2: Package the class in a jar and deploy it to DAS

The custom UDAF class you created should be packaged as a jar and copied to the `<DAS_HOME/repository/components/lib` directory of each WSO2 DAS node in the cluster.

Step 3: Update the Spark UDF configuration file

Add the newly created custom UDAF to the `<DAS_HOME>/repository/conf/analytics/spark/spark-udf-config.xml` file as shown in the example below.

```

<udf-configuration>
  <custom-udaf-classes>
    <custom-udaf>
      <alias>geometricMean</alias>
      <class-name>com.whitesnake.analytics.udaf.GeometricMeanUDAF</class-name>
    </custom-udaf>
  </custom-udaf-classes>
</udf-configuration>

```

This configuration is required for Apache Spark to identify and use the newly defined custom UDAF.

The `alias` parameter denotes the name of the UDAF that should be used within Spark analytics scripts, and the `class-name` parameter points to the fully-qualified name of the implementation class.

This pattern allows you to deploy UDAFs to a WSO2 DAS instance without any configuration changes.

Step 1: Create the POJO class

The following is an example of a UDAF POJO for the `geometricMean` custom UDAF class.

```

package com.whitesnake.analytics.udaf.carbon;

import org.apache.spark.sql.Row
import org.apache.spark.sql.expressions.MutableAggregationBuffer
import org.apache.spark.sql.types._
import org.wso2.carbon.analytics.spark.core.udf.CarbonUDAF

/**
 * This UDAF could be used to calculate the geometric mean of the given set of
numbers.
 *
 * The implementation extends the CarbonUDAF abstract class, and so could be deployed
 * to DAS through OSGi.
 */
class CarbonGeometricMeanUDAF extends CarbonUDAF {

    override def getAlias: String = "geometricMean"

    override def inputSchema: StructType =
StructType(StructField("value", DoubleType) :: Nil)

    override def bufferSchema: StructType = StructType(
StructField("count", LongType) :: StructField("product", DoubleType) :: Nil
)

    override def dataType: DataType = DoubleType

    override def deterministic: Boolean = true

    override def initialize(buffer: MutableAggregationBuffer): Unit = {
buffer(0) = 0L
buffer(1) = 1.0
}

    override def update(buffer: MutableAggregationBuffer, input: Row): Unit = {
buffer(0) = buffer.getAs[Long](0) + 1
buffer(1) = buffer.getAs[Double](1) * input.getAs[Double](0)
}

    override def merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit = {
buffer1(0) = buffer1.getAs[Long](0) + buffer2.getAs[Long](0)
buffer1(1) = buffer1.getAs[Double](1) * buffer2.getAs[Double](1)
}

    override def evaluate(buffer: Row): Any = {
math.pow(buffer.getDouble(1), 1.toDouble / buffer.getLong(0))
}
}

```

This class implements the functionality of the Spark `org.apache.spark.sql.expressions.UserDefinedAggregateFunction` abstract class, and therefore, the functionality of the following should be implemented.

- **inputSchema:** The schema of the input rows.
- **bufferSchema:** The schema of intermediate results.

- `dataType`: The datatype of the final result.
- `Deterministic`: This denotes whether the same inputs always produce the same results.
- `initialize()`: This is called once per node for a given group.
- `update()`: This is called once per input record.
- `merge()`: This is called to compute partial results and combine them together.
- `evaluate()`: This is called to compute the final result.
- `getAlias()`: This denotes what the alias of the custom UDAF should be, at runtime. The return value of this method should be the same as the one used to call up the custom UDAF in Spark scripts.

For more information on each of the above parameters, see the [official Spark Javadoc on UDAFs](#).

Step 2: Package the class as an OSGi bundle and deploy to DAS

Instead of deploying the UDAF class as a jar, you should package it as an OSGi bundle and copy it to the `<DAS_HOME>/repository/components/dropins` directory of each WSO2 DAS node in the cluster.

Spark Troubleshooting

Page under construction.

Apache Spark provides a set of user interfaces (UI) that allow you to monitor and troubleshoot the issues in a Spark cluster. This section helps you to understand the information accessed from these UIs.

The following are the default ports of the main UIs available for Spark. These ports can be configured in the `<DAS_HOME>/repository/conf/analytics/spark/spark-defaults.conf` file.

These ports are only available when WSO2 DAS is deployed as a Spark cluster.

UI	Default port
Master UI**	8081
Worker UI**	1150
Application UI	4040

Master UI

Incremental Processing

Incremental processing is a processing method which involves processing only a data partition newly added to a dataset when the existing data is already processed, instead of re-processing the complete dataset. This processing method improves efficiency by eliminating the system overhead of re-processing already processed data.

e.g., The data in an Analytics table needs to be summarized per day. The following table illustrates how the dataset is processed when the summarization script is run.

	Without Incremental Processing	With Incremental Processing
1st Run	The complete dataset is processed to summarize the data.	The complete dataset is processed to summarize the data.
2nd Day	The complete dataset is processed to summarize the data.	Only the updates made to the dataset during the previous day are processed to summarize the data.

Once the summarization is complete, you need to commit the status to indicate that the data was successfully processed. This is done via the following command.

`INCREMENTAL_TABLE_COMMIT orders;`

Publishing events

Incremental analytics uses the timestamps of the events sent when retrieving the data for processing.

When defining event streams for incremental analytics, you can add an extra attribute to the event payload named `_timestamp` of the `LONG` attribute type. This allows a specific timestamp to be specified for each event. For more information, see [Understanding Event Streams and Event Tables - Adding an event stream](#).

If the `_timestamp` attribute is not specified in the event stream definition, the timestamp of each event is derived from the system date at the time the event was persisted in the database.

Syntax

```
create temporary table orders using CarbonAnalytics options (tableName "ORDERS",
schema "customerID STRING, phoneType STRING, OrderID STRING, cost DOUBLE, _timestamp
LONG -i", incrementalParams "orders, DAY");
```

In order to apply incremental processing to an Analytics table, the `incrementalParams` attribute should be added to the table definition as shown in the extract above. If this parameter is not added, the table is considered a typical analytics table, and the complete table is processed for each query.

The `incrementalParams` attribute should be added to the definition of the table from which the data is read.

Parameters

The following parameters are configured to support incremental processing as shown in the above sample syntax.
Unique ID

Name	uniqueID
Required/Optional	Required
Description	This is the unique ID of the incremental analytics definition. This ID should be used in the incremental table commit command as shown in the sample syntax.

Time Period

Name	timePeriod
Required/Optional	Required

Description	<p>The duration of the time period that you are processing. This can be MINUTE, HOUR, DAY, MONTH or YEAR. DAS has the ability to process the timestamp of each event and identify the unit of time during which it was sent.</p> <p>e.g., If you specify HOUR as the time period, an event sent at 10.20PM is identified as an event sent during the 22.00 – 23.00 hour.</p>																																										
Example	<p>The following is a list of events, and it is required to calculate the number of orders placed during each day.</p> <table border="1" data-bbox="394 439 1334 777"> <thead> <tr> <th>Customer ID</th><th>Phone Type</th><th>Order ID</th><th>Cost</th><th>_timestamp</th></tr> </thead> <tbody> <tr> <td>1</td><td>Nexus 5x</td><td>33s1sa2s</td><td>400</td><td>26th May 2016 12:00:01</td></tr> <tr> <td>12</td><td>Galaxy S7</td><td>kskds221</td><td>600</td><td>27th May 2016 02:00:02</td></tr> <tr> <td>43</td><td>iPhone 6s</td><td>sadl3122</td><td>700</td><td>27th May 2016 15:32:04</td></tr> <tr> <td>2</td><td>MoTo X</td><td>sdda221s</td><td>350</td><td>27th May 2016 16:22:10</td></tr> <tr> <td>32</td><td>LG G5</td><td>lka2s24dkQ</td><td>550</td><td>27th May 2016 19:42:42</td></tr> </tbody> </table> <p>The summarization script is run on 27th May 2016, at 12.00 noon. The last processed order ID kskds221.</p> <p>Then the summarization table is as shown below.</p> <table border="1" data-bbox="394 931 786 1110"> <thead> <tr> <th>Day</th><th>Event Count</th></tr> </thead> <tbody> <tr> <td>26th May 2016</td><td>1</td></tr> <tr> <td>27th May 2016</td><td>1</td></tr> </tbody> </table> <p>As a result, the summarized table for 27th May 2016 should have 1event (because the other events that arrived on 27th May 2016 were received after 12.00 noon).</p> <p>The next time the script is run, WSO2 DAS checks the timestamp of the event that was processed last. In this example, the last processed event is order ID kskds221 with timestamp 27th May 2016 02:00:02. Then DAS retrieves the data from the time period starting at 27th May 2016 00:00:00 to process all the data that has a timestamp greater than 27th May 2016 00:00:00.</p> <p>This updates the value for the entry 27th May 2016 with the value 4 in the summarization table. The summarization table is not affected by any data received before 26th May 2016 because no data is retrieved from that time period. The resulting summarization table is shown below.</p> <table border="1" data-bbox="394 1607 786 1786"> <thead> <tr> <th>Day</th><th>Event Count</th></tr> </thead> <tbody> <tr> <td>26th May 2016</td><td>1</td></tr> <tr> <td>27th May 2016</td><td>4</td></tr> </tbody> </table>	Customer ID	Phone Type	Order ID	Cost	_timestamp	1	Nexus 5x	33s1sa2s	400	26th May 2016 12:00:01	12	Galaxy S7	kskds221	600	27th May 2016 02:00:02	43	iPhone 6s	sadl3122	700	27th May 2016 15:32:04	2	MoTo X	sdda221s	350	27th May 2016 16:22:10	32	LG G5	lka2s24dkQ	550	27th May 2016 19:42:42	Day	Event Count	26th May 2016	1	27th May 2016	1	Day	Event Count	26th May 2016	1	27th May 2016	4
Customer ID	Phone Type	Order ID	Cost	_timestamp																																							
1	Nexus 5x	33s1sa2s	400	26th May 2016 12:00:01																																							
12	Galaxy S7	kskds221	600	27th May 2016 02:00:02																																							
43	iPhone 6s	sadl3122	700	27th May 2016 15:32:04																																							
2	MoTo X	sdda221s	350	27th May 2016 16:22:10																																							
32	LG G5	lka2s24dkQ	550	27th May 2016 19:42:42																																							
Day	Event Count																																										
26th May 2016	1																																										
27th May 2016	1																																										
Day	Event Count																																										
26th May 2016	1																																										
27th May 2016	4																																										

The following sample further demonstrates how the incremental processing is carried out.

Click here to view the complete sample.

An event stream is defined with the following configuration. For more information about event streams, see [Understanding Event Streams and Event Tables](#).

```
{  
    "streamId": "APIStats:1.0.0",  
    "name": "APIStats",  
    "version": "1.0.0",  
    "nickName": "",  
    "description": "",  
    "metaData": [],  
    "correlationData": [],  
    "payloadData": [  
        {  
            "name": "name",  
            "type": "STRING"  
        },  
        {  
            "name": "count",  
            "type": "INT"  
        },  
        {  
            "name": "_timestamp",  
            "type": "LONG"  
        }  
    ]  
}
```

The Spark script written to process the data received by this event stream is as follows.

```

create temporary table APIStats using CarbonAnalytics options (tableName "APIStats",
schema "name STRING, count INT, _timestamp LONG", incrementalParams "api_stats_1,
MINUTE");

create temporary table APIStatsMinuteSummary using CarbonAnalytics options
(tableName "APIStatsMinSummary", schema "name STRING, count INT, _timestamp LONG,
year INT, month INT, day INT, hour INT, min INT", primaryKeys "name, min, hour, day,
month, year", incrementalParams "api_stats_min_1, HOUR");

create temporary table APIStatsHourSummary using CarbonAnalytics options (tableName
"APIStatsHourSummary", schema "name STRING, count INT, _timestamp LONG, year INT,
month INT, day INT, hour INT", primaryKeys "name, hour, day, year",
incrementalParams "api_stats_hour_1, DAY");

create temporary table APIStatsDaySummary using CarbonAnalytics options (tableName
"APIStatsDaySummary", schema "name STRING, count INT, _timestamp LONG, year INT,
month INT, day INT", primaryKeys "name, day, year");

insert into table APIStatsMinuteSummary select name, sum(count) as count,
getMinuteStartingTime(getYear(first(_timestamp)), getMonth(first(_timestamp)),
getDay(first(_timestamp)), getHour(first(_timestamp)), getMinute(first(_timestamp))) as
_timestamp, getYear(first(_timestamp)) as year, getMonth(first(_timestamp)) as
month, getDay(first(_timestamp)) as day, getHour(first(_timestamp)) as hour,
getMinute(first(_timestamp)) as min from APIStats group by name,
getYear(_timestamp), getMonth(_timestamp), getDay(_timestamp), getHour(_timestamp),
getMinute(_timestamp)

INCREMENTAL_TABLE_COMMIT api_stats_1;

insert into table APIStatsHourSummary select name, sum(count) as count,
getHourStartingTime(year, month, day, hour) as _timestamp, year, month, day, hour
from APIStatsMinuteSummary group by name, year, month, day, hour

INCREMENTAL_TABLE_COMMIT api_stats_min_1;

insert into table APIStatsDaySummary select name, sum(count) as count,
getDateStartingTime(year, month, day) as _timestamp, year, month, day from
APIStatsHourSummary group by name, year, month, day

INCREMENTAL_TABLE_COMMIT api_stats_hour_1;

```

The `incrementalParams "api_stats_1, MINUTE"` parameter specifies incremental processing to be applied. When the script is run, the system identifies the last minute during which the summarization was down, and processes all events with a timestamp that is greater than the timestamp of the start of that minute.

To use the above sample, you need to add [this UDF](#).

For detailed instructions to add a UDF, see [Creating Spark User Defined Functions](#).

Monitoring Spark Performance

Predictive Analytics

WSO2 DAS allows you to carry out deploy datasets and generate predictive models via the Machine Learner Wizard. Follow the procedure below to access the Machine Learner Wizard.

1. Access the WSO2 DAS Management Console using the following URL, and log in with your username and password.
`https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. In the **Main** tab, click **Machine Learner Wizard**.

For detailed information about using this wizard, see the [WSO2 ML User Guide](#).

WSO2 DAS 3.1.0 and WSO2 ML 1.1.1 use Apache Spark 1.6.2 whereas the previous versions use Spark 1.4.1. Due, when you migrate to ML 1.1.1 or DAS 3.1.0 from a previous version, some of the models will not be backward compatible depending on the algorithm type used.

For information of the algorithms that do not support backward compatibility when models are migrated to a WSO2 DAS/WSO2 ML version using Apache Spark 1.6.2, see [Machine Learner Algorithms](#).

Using a ML Model Within WSO2 CEP

For information on using a ML model (which you generated using WSO2 ML) within WSO2 CEP, go to [WSO2 CEP Extension for ML Predictions](#).

Using a ML Model Within WSO2 ESB

For information on using a ML model (which you generated using WSO2 ML) within WSO2 ESB, go to [Predict Mediator for WSO2 ESB](#).

Communicating Results

After collecting data, and performing various analysis on them to produce meaningful information, the final step in [business activity monitoring](#) is to present this information. WSO2 DAS queries the analyzed information and shows it in various UIs.

The following sections describe how to work with DAS components to query and present analyzed information:

- [Visualizing Results](#)
- [Creating Alerts](#)
- [Communicating Results Through REST API](#)
- [Analytics JavaScript \(JS\) API](#)

Visualizing Results

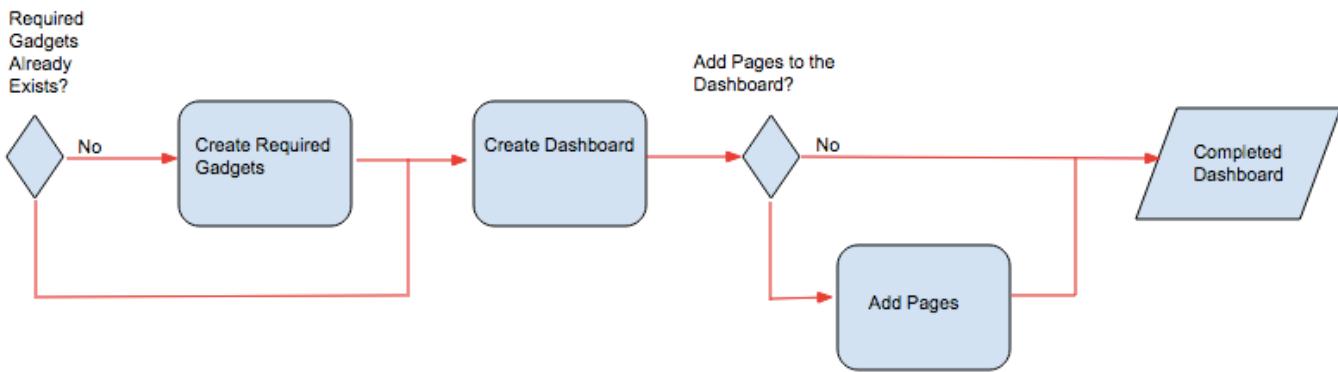
Following sections describe the methods that are available to visualize results in WSO2 DAS.

- [Analytics Dashboard](#)
- [Geo Dashboard](#)

Analytics Dashboard

Note that this dashboard is not supported with IBM JDK.

Analytics dashboard application is the data visualization component of WSO2 Data Analytics Server. You can create dashboards and real time gadgets event streams. A gadget displays the real-time information processed by WSO2 DAS 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 DAS 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.

Service Provider	Prerequisites
Batch Data Source	A persisted event stream should exist in order to populate the gadget with persisted data. To define an event stream, see Understanding Event Streams and Event Tables . To persist an event stream, see Persisting Data for Batch Analytics .
Relational Database Source	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 and Event Tables . • 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 DAS Analytics Dashboard using the following URL.
`https://<HOST_NAME>:<PORT>/portal/dashboard`
2. Enter your username and password, and click **Login**.

Login

The screenshot shows the WSO2 Data Analytics Server login interface. It consists of three input fields: a top field containing 'admin', a middle field containing 'admin' with the placeholder 'Password' visible, and a bottom blue button labeled 'Login'.

You can deploy a dashboard and/or its components (e.g. layouts, gadgets, and widgets) in the Analytics Dashboard of WSO2 DAS by bundling them as artifacts of a Carbon Application (cApp). For instructions on deploying cApps in WSO2 DAS, see [Packaging Artifacts as a C-App Archive](#).

Step 2: Create required gadgets

Follow the procedure below to create a gadget to display information processed by WSO2 DAS as required.

1. In the Analytics Dashboard, click the menu icon and then click **Gadgets** to open the **Gadgets** page as demonstrated below

The screenshot shows the Analytics Dashboard header with a 'CREATE DASHBOARD' button and a menu icon. Below the header, there are three main categories: DASHBOARDS, GADGETS, and LAYOUTS. The GADGETS category is highlighted with a white background and black text.

Dashboards

A card for the 'Geo Dashboard'. It displays the title 'Geo Dashboard' and the URL 'URL: geo-dashboard'. At the bottom, there are three buttons: 'View' (with a circular icon), 'Design' (with a pencil icon), and 'Settings' (with a gear icon).

2. Click **GENERATE GADGET** to open the **Generate a Gadget** wizard.
3. Enter information as follows to configure a gadget.

The screenshot shows a three-step wizard titled "Generate a Gadget". The current step is "1 Select Provider", which is highlighted in blue. The previous step, "2 Configure Provider", and the next step, "3 Configure Chart", are shown in grey. At the top left is a "CANCEL" button with a close icon. Below the steps is a section labeled "Select Provider *". A dropdown menu is open, showing "Batch Data Source" as the selected option. There are up and down arrows to change the selection.

- 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, **Batch Data Source** is selected to publish data updated in an event table.

Provider	Source of Data Published in Gadget
Batch Data Source	<p>An event table in DAS that contains persisted data from an event stream.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> For more information about events streams and event data, see Understanding Event Streams and Event Tables. </div>
Relational Database Source	A datasource that can connect to an RDBMS database.

Realtime Data Source	<p>Real-time events received in event streams.</p> <p>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.</p>
REST Data Source	A REST endpoint

- b. In the **Event Table** field, select the event stream from which the events to be published are derived. Then click **Next**. In this example, a table named CITY_USAGE is selected.
- c. Configure a chart as follows.

Parameter Name	Description	Example
Gadget Name	The text to be displayed as the name of the gadget.	USAGE - CITIES
Select Chart Type	The type of chart that you want the gadget to display.	Bar Chart
X-Axis	Select the attribute for the X axis of the bar chart.	metro_area
Y-Axis	Select the attribute for the Y axis of the bar chart.	avg_usage
Color domain	The points mapped on the line chart will be differentiated using colour based on the value of the attribute selected for this parameter.	metro_area
Max length	The maximum length of the chart in terms of the number of records displayed at a given time.	30

- d. Click **Preview** to see the preview of the gadget in the lower section of the page.
- e. 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 <DAS_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](#).

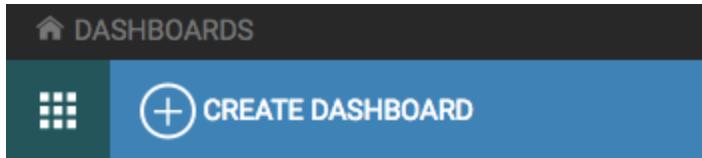
Step 3: Create a new dashboard

If you wish to create a dashboard based on a custom theme, see [Adding a Custom Theme for a 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.



Dashboards

power-dashboard
URL: power-dashboard

Power

View Design Settings

2. Click **CREATE DASHBOARD** in the top navigator to open the **CREATE A DASHBOARD** page.
3. Enter information as follows.



Dashboards

No dashboards found.

To create a dashboard click [here](#)

- a. Enter a name and a description for the dashboard. In this example, the name and the description is

entered as follows.

Parameter	Value
Name of your Dashboard	Smart Home
Description	Indicates the average power usage for different cities.

- b. 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.
- c. Click the icon for gadgets. Drag and drop a gadget displayed into the required column of your dashboard. In this example, the Usage - Cities 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](#).

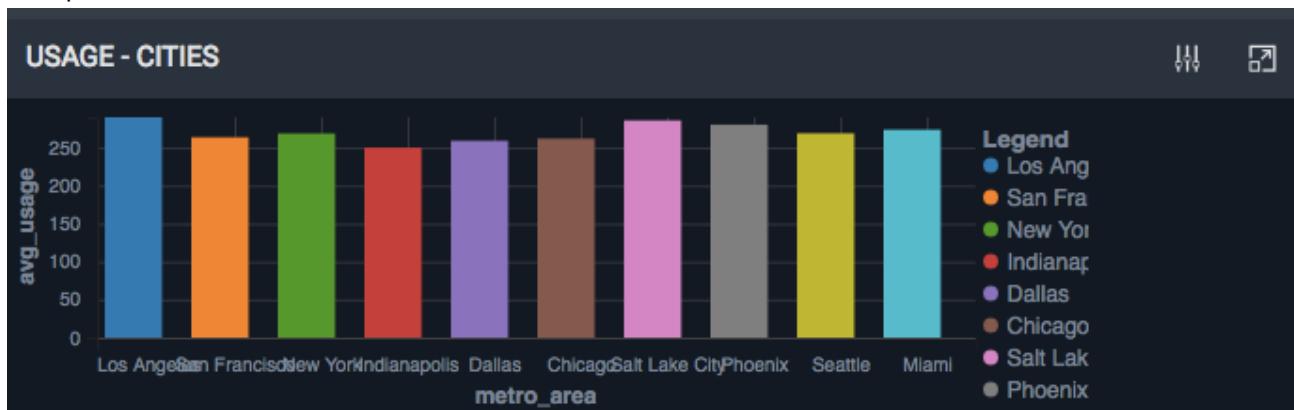
Step 4: View information

Follow the procedure below to view information in a dashboard in the WSO2 DAS Analytics Dashboard.

1. Access the WSO2 DAS 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.

In this example, data is simulated for the gadget created by building the `<DAS_HOME>/samples/smart-home/resources/access.log` file. For more information, see [Analyzing Smart Home Data](#).

The gadget is updated as shown in the following image of the USAGE - CITIES gadget created in this example.



Geo Dashboard

The Geo Dashboard of WSO2 DAS 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.

Alert type	Description
Speed alert	You can specify a maximum speed limit to all spatial objects. If an object exceeds the specified speed, an alert will be generated.
Proximity alert	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.
Within alert	You can specify a geo area and if any spatial object comes in to the specified area an alert will be generated.
Stationery alert	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.
Congestion Alert	You can define a geo area. An alert is generated when the congestion level in that area changes.

- Prerequisites
- Running the Geo Dashboard
- Executing the producer
- Accessing the Geo Dashboard

You can use the Geo Dashboard of WSO2 DAS 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 <DAS_HOME>/samples/cep/utils/geo-dashboard-extensions/ directory to the <DAS_HOME>/repository/components/lib/ directory.

Running the Geo Dashboard

Navigate to the <DAS_HOME>/repository/resources/geo-dashboard directory and copy all the folders into the <DAS_HOME>/repository/deployment/server directory.

Executing the producer

Follow the steps below to run the tfl-feed producer client from the command line.

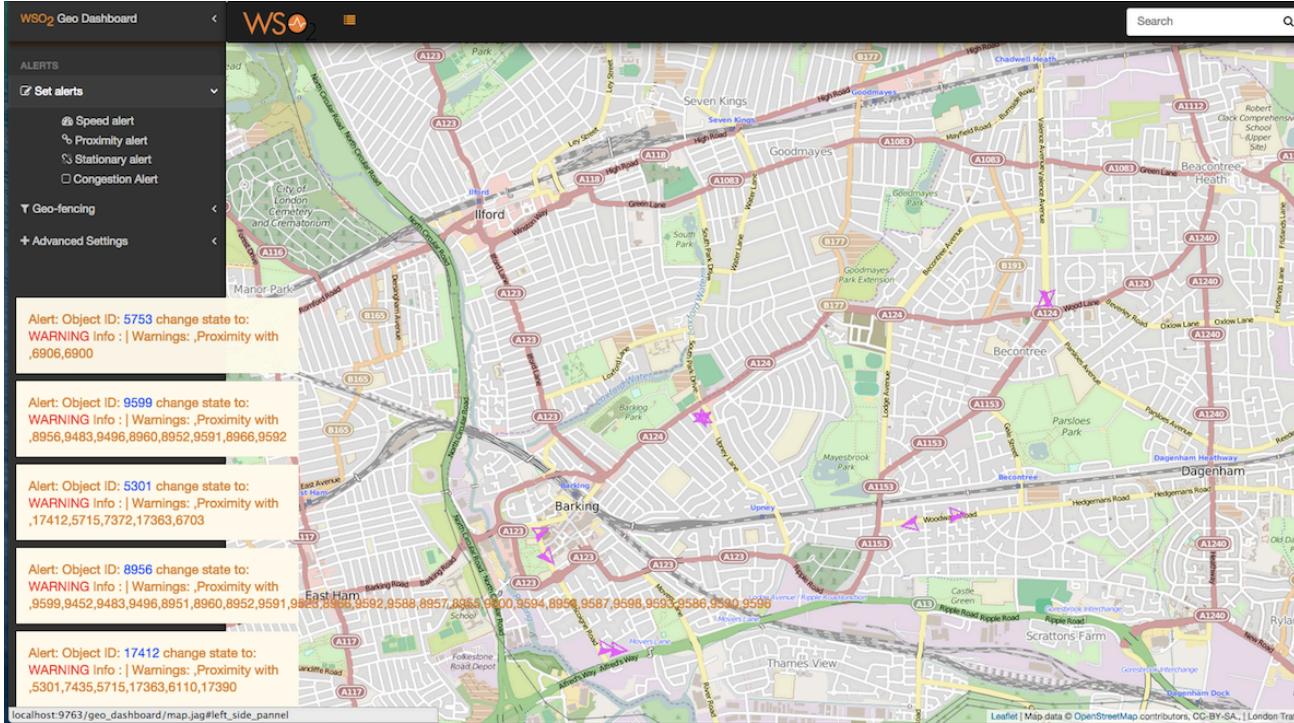
1. Download [GeoTools GIS toolkit](#)
2. Copy the following client JAR files from downloaded <GeoTools_HOME> directory to both <DAS_HOME>/repository/components/lib/ directory and <DAS_HOME>/samples/cep/lib directory.
 - gt-geojson-13.1.jar
 - gt-main-13.1.jar
 - gt-metadata-13.1.jar
 - jai_core-1.1.3.jar
 - jai_imageio-1.1.jar
3. Download [JTS Topology Suite](#).
4. Copy jts-1.8.jar from <JTS_HOME>/lib directory to both <DAS_HOME>/repository/components/lib directory and <DAS_HOME>/samples/cep/lib directory.

5. Navigate to <DAS_HOME>/samples/cep/producers/tfl-feed/ directory, and execute the ant command in a new tab of your DAS 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 DAS server. For instructions, see [Running the Product](#).
2. Access the Geo Dashboard application in your Web browser using the following URL: <https://localhost:9443/portal/dashboards/geo-dashboard>



This dashboard can be used to define alerts, define geo fences, click on spatial objects to view information relating to them etc.

Creating 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.

- [Configuring global properties](#)
- [Event publisher configuration](#)
- [Creating event publishers](#)
- [Enabling statistics for event publishers](#)
- [Enabling tracing for event publishers](#)
- [Deleting event publishers](#)
- [Editing event publishers](#)

Configuring global properties

Global properties can be set for individual output event adapter types in the <DAS_HOME>/repository/conf/output-event-adapters.xml file. A global property set for an output event adapter type in this file applies to all the publishers with that adapter type. If a property available for an adapter type by default is removed, the default value of the property applies. Click the relevant tab to view the properties available by default for a specific output event adapter type.

Custom properties cannot be added as global properties.

RDBMSHTTPJMSMQTTKAFKAEmailUIWebsocket-localWebsocketSOAP

When the output event adapter type is RDBMS, it is allowed to change the queries used to perform the standard database operations. This enables you to use RDBMS database types that use different queries. Customised values can be defined for the following used in standard queries.

Attribute/activity	Current query
string	VARCHAR(255)
double	DOUBLE
integer	INT
long	BIGINT
float	FLOAT
createTable	CREATE TABLE \$TABLE_NAME (\$COLUMN_TYPES)
insertDataToTable	INSERT INTO \$TABLE_NAME (\$COLUMNS) VALUES (\$VALUES)
isTableExist	SELECT * FROM \$TABLE_NAME limit 1
updateTableRow	UPDATE \$TABLE_NAME SET \$COLUMN_VALUES WHERE \$CONDITION
comma	,
questionMark	?
equal	=
and	AND
selectAllColumnsDataTypeInTable	SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '\$TABLE_NAME'
selectFromTable	SELECT \$COLUMNS FROM \$TABLE_NAME
oracle.string	varchar2(255)
oracle.long	CLOB
oracle.double	BINARY_DOUBLE
oracle.isTableExist	SELECT * FROM \$TABLE_NAME WHERE ROWNUM = 1

oracle.selectAllColumnsDataTypeInTable	SELECT COLUMN_NAME, DATA_TYPE FROM USER_TAB_COLS WHERE TABLE_NAME = '\$TABLE_NAME'
mssql.string	varchar2(255)
mssql.isTableExist	SELECT TOP 1 * FROM \$TABLE_NAME
mssql.selectAllColumnsDataTypeInTable	SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '\$TABLE_NAME'
h2.integer	varchar2(255)
h2.long	REAL
h2.selectAllColumnsDataTypeInTable	SHOW COLUMNS FROM \$TABLE_NAME

The following properties are available for the `http` output event adapter type.

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 size of the queue that is used to hold events before they are forwarded to the event stream.	Integer	10000
defaultMaxConnectionsPerHost	The maximum number of connections allowed per host configuration.	Integer	50
maxTotalConnections	The maximum number of connections allowed overall.	Integer	1000

The following properties are available for the `jms` output event adapter type.

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

The following properties are available for the `mqtt` output event adapter type.

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
connectionKeepAliveInterval	The time interval in milliseconds at which a check should be carried out to identify inactive threads.	Integer	60

The following properties are available for the `kafka` output event adapter type.

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
--------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------	-------

The following properties are available for the email output event adapter type.

Property Key	Description	Data Type	Default Value
mail.smtp.from	The email address used by the publisher to publish events.	String	abcd@gmail.com
mail.smtp.user	The username used by the publisher to publish events via email.	String	abcd
mail.smtp.password	The password used by the publisher to publish events via email.	String	xxxx
mail.smtp.host	The host of the email server.	String	smtp.gmail.com
mail.smtp.port	The port of the email server.	Integer	587
mail.smtp.starttls.enable	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.	Boolean	true
mail.smtp.auth	This property specifies whether SMTP authentication is enabled or not.	Boolean	true
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

The following properties are available for the UI output event adapter type.

Property Key	Description	Data Type	Default Value
eventQueueSize	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.	Integer	30
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

The following properties are available for the websocket-local output event adapter type.

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

The following properties are available for the websocket output event adapter type.

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

The following properties are available for the soap output event adapter type.

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
axis2ClientConnectionTimeout	The number of milliseconds allowed to elapse before the Axis2 client connection times out.	Integer	10000
reuseHTTPClient	If this property is set to true, it is allowed to reuse the connection to the HTTP client for subsequent requests.	Boolean	true
autoReleaseConnection	If this property is set to true, inactive connections are automatically killed.	Boolean	true
maxConnectionsPerHost	The maximum number of connections allowed per host configuration.	Integer	50

Event publisher configuration

An event publisher configuration has four main sections as follows.

Create a New Event Publisher

The screenshot shows the 'Enter Event Publisher Details' form. It includes sections for 'Event Publisher Name*', 'From', 'To', 'Static Adapter Properties', 'Mapping Configuration', and 'Advanced'. The 'From' section is currently active, showing 'testEventStream:1.0.0' selected as the event source. The 'To' section shows 'websocket' selected as the output adapter type. The 'Mapping Configuration' section shows 'text' selected as the message format. A red box highlights the 'From' section, and another red box highlights the 'Advanced' link.

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 .

```
<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.

Section	Description
From	The event stream from which the event publisher will fetch the events for publishing.

To	An output event adapter(transport) configuration that is used to send the events to.
Adapter properties	<p>Output event adapters contain three types of adapter properties in their configuration as explained below.</p> <ul style="list-style-type: none"> 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. <code>http://localhost:8000/endpoint/{{endpointId}}</code> 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.

- To create an event publisher via the management console, you must at least have one event stream defined.
- Once an event publisher is created and saved, its XML configuration is saved in the <DAS_HOME>/repository/deployment/server/eventpublishers directory.

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 Publishers** list as shown below.

Home > Manage > Event > Publishers Help

Available Event Publishers

[Add Event Publisher](#)

1 Active Event Publishers. 0 Inactive Event Publishers (All)

Event Publisher Name	Message Format	Output Event Adapter Type	Input Stream ID	Actions
TestEventPublisher	text	websocket-local	Test Stream:1.0.0	

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 should start with `<eventPublisher>` as the root element.

In the following configuration, specify the respective adapter properties based on the transport type of the publisher within the `<from>` element. For the respective adapter properties of the event publisher configuration based on the transport type, see [Publishing Events via Various Transports](#).

```
<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.

Adapter property	Description
name	Name of the event publisher
statistics	Whether monitoring event statistics is enabled for the publisher
trace	Whether tracing events is enabled for the publisher
xmlns	XML namespace for event receivers
streamName	Name of the event stream from which the publisher publishes events.
version	Version of the event stream from which the publisher publishes events.
customMapping	Whether a custom mapping is enabled on the receiver.
type	Type of the enabled custom mapping.
eventAdapterType	Type of the event adapter.

2. Add the XML file to the `<DAS_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.

Home > Manage > Event > Publishers

Inactive Event Publishers

File Name	Reason for being inactive	Actions
TestCassandraPublisher.xml	Deployment exception: Invalid XML for file TestCassandraPublisher.xml	Delete Source View

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.

For more information on monitoring event statistics of event publishers, see [Event Statistics](#).

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.

For more information on monitoring event statistics of event publishers, see [Event Tracer](#).

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 Configuration** 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](#)
- [Building Custom Event Publishers](#)

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 [Creating Alerts](#).

Configuring adapter properties

Specify the **Static Adapter Properties**, when creating a Cassandra event publisher using the Management Console as shown below.

[Create a New Event Publisher](#)

Enter Event Publisher Details

Event Publisher Name*	CassandraOutputEventAdapter Enter a unique name to identify Event Publisher
From	Test Stream:1.0.0 The stream of events that need to be published sensor_id int
Stream Attributes	
To	cassandra Select the type of Adapter to publish events
Static Adapter Properties	
Hosts *	localhost Hostnames or ipaddresses separated by comma e.g., testhost1,testhost2
Port	9160 The cassandra port, if not defined default port will be used
User Name	admin
Password	*****
Keyspace Name *	CEP_KS
Column Family Name *	CF_Transactions
Strategy Class	SimpleStrategy The strategy of the keyspace, if not defined 'org.apache.cassandra.locator.SimpleStrategy' will be used
Replication Factor	3 The replication factor of keyspace, if not defined '1' will be used
Indexed Columns	group_id Columns to be indexed, separated by comma e.g., key1,key2. Index of type "KEYS" with name "[keyspaceName]_[columnFamilyName]_[columnKey]_Index" will be applied to the columns
Mapping Configuration	
Message Format*	map 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="CassandraOutputEventAdapter" statistics="disable"
trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    .....
    <to eventAdapterType="cassandra">
        <property name="port">9160</property>
        <property name="indexed.columns">key1,key2</property>
        <property name="key.space.name">CEP_KS</property>
        <property name="user.name">admin</property>
        <property name="column.family.name">CF_Transactions</property>
        <property name="hosts">testhost1,testhost2</property>
        <property name="replication.factor">3</property>
        <property encrypted="true" name="password">kuv2MubUUveMyv6GeHrXr9il59ajJIqUI4eoYHcgGKf/BBFOWn96NTjJQI+wYbwjkW6r79
S7L7ZzgYeWx7DlGbff5X3pBN2Gh9yV0BHP1E93QtFqR7uTWi141Tr7V7ZwScwNqJbiNoV+vyLbsqKJE7T3nP8I
h9Y6omygbcLcHzg=</property>
        <property name="strategy.class">SimpleStrategy</property>
    </to>
</eventPublisher>

```

The above adapter properties are described below.

Adapter Property	Description	Configuration file property	Example
Hosts	Hostnames or IP addresses separated by commas	hosts	testhost1,testhost2
Port	The Cassandra port. If you do not define this, the default port will be used	port	9160
User Name	Username for the database	user.name	admin
Password	Password for the database	password	password
Keyspace Name	Cassandra keyspace name	key.space.name	CEP_KS
Column Family Name	Column family namespace under the defined keyspace	column.family.name	CF_Transactions
Strategy Class	The strategy of the keyspace. If you do not define this, org.apache.cassandra.locator.SimpleStrategy will be used	strategy.class	SimpleStrategy
Replication Factor	The replication factor of the keyspace. If you do not define this, 1 will be used.	replication.factor	3
Indexed Columns	Columns to be indexed, separated by commas. Index of type "KEYS" with the name {keyspaceName}_{columnFamilyName}_{columnKey}_Index will be applied to the columns.	indexed.columns	key1,key2

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.

```
<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>
    <!-- Thread Pool Related Properties -->
    <property key="minThread">8</property>
    <property key="maxThread">100</property>
    <property key="keepAliveTimeInMillis">20000</property>
    <property key="jobQueueSize">10000</property>
</adapterConfig>
```

- In gmail [account security settings](#) you may have to enable the Allow less secure apps option in order to connect the 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 [Creating Alerts](#).

Configuring adapter properties

Specify the **Dynamic Adapter Properties**, when creating an email event publisher using the management console as shown below.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name*	EmailOutputEventAdapter ⑦ Enter a unique name to identify Event Publisher						
From	Test Stream:1.0.0 ⑦ The stream of events that need to be published						
Event Source*	sensor id int						
Stream Attributes							
To	email ⑦ Select the type of Adapter to publish events						
Dynamic Adapter Properties <table border="1"> <tr> <td>Email Address*</td> <td>user@gmail.com ⑦ Register publisher for multiple email IDs' separated by ","</td> </tr> <tr> <td>Subject*</td> <td>This is a test mail.</td> </tr> <tr> <td>Email Type</td> <td>text/plain ⑦ Select the email format to be sent</td> </tr> </table>		Email Address*	user@gmail.com ⑦ Register publisher for multiple email IDs' separated by ","	Subject*	This is a test mail.	Email Type	text/plain ⑦ Select the email format to be sent
Email Address*	user@gmail.com ⑦ Register publisher for multiple email IDs' separated by ","						
Subject*	This is a test mail.						
Email Type	text/plain ⑦ Select the email format to be sent						
Mapping Configuration <table border="1"> <tr> <td>Message Format*</td> <td>text ⑦ Select the output message format</td> </tr> <tr> <td colspan="2"> Advanced </td> </tr> </table>		Message Format*	text ⑦ Select the output message format	Advanced			
Message Format*	text ⑦ 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="EmailOutputEventAdapter" 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.

Adapter Property	Description	Configuration file property	Example
Email Address	Email address of the client. Register the publisher for multiple email IDs' by separating them with commas.	email.address	user@gmail.com
Subject	Subject of the email to be sent to the defined email address.	email.subject	This is a test mail.
Email Type	The email format to be sent to the defined email address. If you select text/html for this parameter, the message body should be in valid HTML.	email.type	text/plain

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 [Creating Alerts](#).

Configuring global properties

The following global properties can be set for the HTTP event publisher type in the `<DAS_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.

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 size of the queue that is used to hold events before they are forwarded to the event stream.	Integer	10000

defaultMaxConnectionsPerHost	The maximum number of connections allowed per host configuration.	Integer	50
maxTotalConnections	The maximum number of connections allowed overall.	Integer	1000

Configuring adapter properties

Specify the **Static** and **Dynamic Adapter Properties**, when creating a HTTP event publisher using the management console as shown below.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name*	HTTPOutputEventAdapter ⑦ Enter a unique name to identify Event Publisher								
From	Test Stream:1.0.0 ⑦ The stream of events that need to be published								
Event Source*	sensor id int								
Stream Attributes									
To	http ⑦ Select the type of Adapter to publish events								
Static Adapter Properties <table border="1"> <tr> <td>Proxy Host</td> <td>localhost ⑦ The proxy server host</td> </tr> <tr> <td>Proxy Port</td> <td>8080 ⑦ The proxy server port</td> </tr> <tr> <td>HTTP Client Method*</td> <td>HttpPost ⑦</td> </tr> </table>		Proxy Host	localhost ⑦ The proxy server host	Proxy Port	8080 ⑦ The proxy server port	HTTP Client Method*	HttpPost ⑦		
Proxy Host	localhost ⑦ The proxy server host								
Proxy Port	8080 ⑦ The proxy server port								
HTTP Client Method*	HttpPost ⑦								
Dynamic Adapter Properties <table border="1"> <tr> <td>URL*</td> <td>http://localhost:8080 ⑦ The target HTTP/HTTPS URL, e.g. "http://yourhost:8080/service"</td> </tr> <tr> <td>Username</td> <td>admin ⑦ HTTP BasicAuth username</td> </tr> <tr> <td>Password</td> <td>***** ⑦ HTTP BasicAuth password</td> </tr> <tr> <td>Headers</td> <td>header1: value1, header2: value2 ⑦ Custom HTTP headers, e.g. "header1: value1, header2: value2"</td> </tr> </table>		URL*	http://localhost:8080 ⑦ The target HTTP/HTTPS URL, e.g. "http://yourhost:8080/service"	Username	admin ⑦ HTTP BasicAuth username	Password	***** ⑦ HTTP BasicAuth password	Headers	header1: value1, header2: value2 ⑦ Custom HTTP headers, e.g. "header1: value1, header2: value2"
URL*	http://localhost:8080 ⑦ The target HTTP/HTTPS URL, e.g. "http://yourhost:8080/service"								
Username	admin ⑦ HTTP BasicAuth username								
Password	***** ⑦ HTTP BasicAuth password								
Headers	header1: value1, header2: value2 ⑦ Custom HTTP headers, e.g. "header1: value1, header2: value2"								
Mapping Configuration <table border="1"> <tr> <td>Message Format*</td> <td>text ⑦ Select the output message format</td> </tr> <tr> <td colspan="2"> Advanced </td> </tr> </table>		Message Format*	text ⑦ Select the output message format	Advanced					
Message Format*	text ⑦ Select the output message format								
Advanced									
<input type="button" value="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

ou 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="HTTPOutputEventAdapter" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventpublisher">
    .....
    <to eventAdapterType="http">
        <property name="http.client.method">HttpPost</property>
        <property name="http.username">admin</property>
        <property name="http.proxy.host">yourhost</property>
        <property name="http.proxy.port">8080</property>
        <property encrypted="true"
name="http.password">kuv2MubUUveMyv6GeHrXr9il59ajJIqUI4eoYHcgGKf/BBFOWn96NTjjQI+wYbWjk
W6r79S7L7ZzgYeWx7DlGbff5X3pBN2Gh9yv0BHP1E93QtFqR7uTWi141Tr7V7ZwScwNqJbiNoV+vyLbsqKJE7T
3nP8Ih9Y6omygbcLcHzg=</property>
        <property name="http.headers">header1: value1, header2: value2</property>
        <property name="http.url">http://localhost:8080</property>
    </to>
</eventPublisher>

```

The above adapter properties are described below.

Static adapter properties

Adapter Property	Description	Configuration file property	Example
Proxy Host	The proxy server host	<code>http.proxy.host</code>	<code>yourhost</code>
Proxy Port	The proxy server port	<code>http.proxy.port</code>	<code>8080</code>
HTTP Client Method	The standard HTTP client method	<code>http.client.method</code>	<code>HttpPost</code>

Dynamic adapter properties

Adapter Property	Description	Configuration file property	Example
URL	The target HTTP/HTTPS URL	<code>http.url</code>	http://yourhost:8080/service or https://yourhost:8080/service
Username	HTTP BasicAuth username	<code>http.username</code>	<code>admin</code>
Password	HTTP BasicAuth password	<code>http.password</code>	<code>admin</code>
Headers	Custom HTTP headers	<code>http.headers</code>	<code>header1: value1, header2: value2</code>

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 `<DAS_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.

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

- 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

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](#).

2. Add the following ActiveMQ JMS-specific JAR files to the `<PRODUCT_HOME>/repository/components/lib/` directory.
 - `<ACTIVEMQ_HOME>/lib/activemq-core-xxx.jar`
 - `<ACTIVEMQ_HOME>/lib/geronimo-j2ee-management_1.1_spec-1.0.1.jar`
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: **topic.{topicName} = {topicName}**

```
connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/test?brokerlist='tcp://localhost:5672'  
topic.topicMap = topicMap  
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 [Creating Alerts](#).

Configuring adapter properties

Specify the **Static** and **Dynamic Adapter Properties**, when creating an ActiveMQ JMS event publisher using the management console as shown below.

Create a New Event Publisher

Enter Event Publisher Details	
Event Publisher Name*	ActiveMQJMSEventPublisher ② Enter a unique name to identify Event Publisher
From	dashboardTest:1.0.0 ② The stream of events that need to be published
Event Source*	x int, y string, z string
Stream Attributes	
To	jms ② Select the type of Adapter to publish events
Static Adapter Properties	
JNDI Initial Context Factory Class*	org.apache.activemq.jndi.ActiveMQInitialContextFactory ② JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.
JNDI Provider URL*	tcp://localhost:61616 ② URL of the JNDI provider.
Username	jms-user
Password	*****
Connection Factory JNDI Name*	TopicConnectionFactory ② The JNDI name of the connection factory.
Destination Type*	topic ② Type of the destination.
Destination*	topicMap allow
Concurrent Publishers	② Concurrent publishers can yield high throughput, but may result in out-of-order message delivery.
Dynamic Adapter Properties	
Header	② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)
Mapping Configuration	
Message Format*	text ② Select the output message format
Advanced	
<input type="button" value="Add Event Publisher"/> <input type="button" value="Test 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.

```

<eventPublisher name="ActiveMQJMSOutputEventAdapter" statistics="disable"
trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    .....
    <to eventAdapterType="jms">
        <property
name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFact
ory</property>
        <property name="java.naming.provider.url">tcp://localhost:61616</property>
        <property name="transport.jms.UserName">jms-user</property>
        <property encrypted="true"
name="transport.jms.Password">ImE/+i4TR0c7p97CWbd8bUgfXfC8XcKWVwIwXxw+ROUFvxOR3+61S6YX
qZK7dkKTLgBBFNmB2czfSiJrUz9jCYxFXSUquCfqFs8UKXx3976sjmM+giTTyPJnyCNilceF2fMPZ0abOJdq7g
D+zi9IeoX14EPnZUuY9sOUOGFg7B8=</property>
        <property name="transport.jms.DestinationType">topic</property>
        <property
name="transport.jms.Header">header_name1:header_value1,header_name2:header_value2</pro
perty>
        <property name="transport.jms.Destination">topicMap</property>
        <property
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    </to>
</eventPublisher>

```

The above adapter properties are described below.

Static adapter properties

Adapter Property	Description	Configuration file property
JNDI Initial Context Factory Class	The JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface	<code>java.naming.factory.initial</code>
JNDI Provider URL	URL of the JNDI provider	<code>java.naming.provider.url</code>
Username	Valid username for the JMS connection	<code>transport.jms.UserName</code>
Password	Valid password for the JMS connection	<code>transport.jms.Password</code>
Connection Factory JNDI Name	The JNDI name of the connection factory	<code>transport.jms.ConnectionFactoryJNDIN</code>
Destination Type	The sort order for messages that arrive on a specific destination	<code>transport.jms.DestinationType</code>
Destination	The topic or queue to which WSO2 CEP sends messages by publishing.	<code>transport.jms.Destination</code>
Concurrent Publishers	If concurrent publishers are allowed to publish events to a JMS broker using multiple threads.	<code>transport.jms.ConcurrentPublishers</code>

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

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`.

```
INITIAL_CONTEXT_FACTORY=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:

```

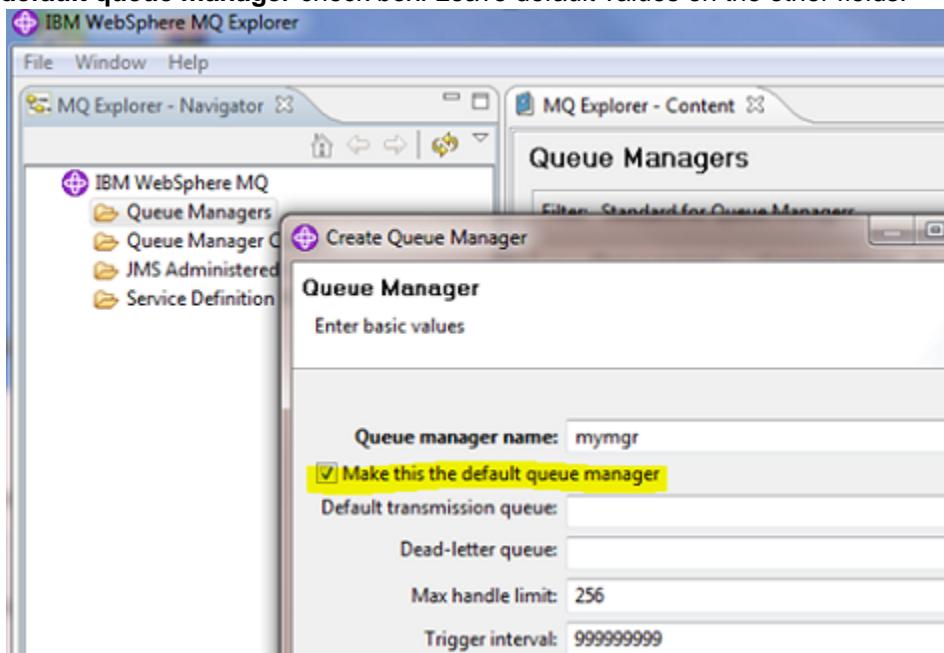
# appropriate one should be uncommented.
#
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
#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
.....

```

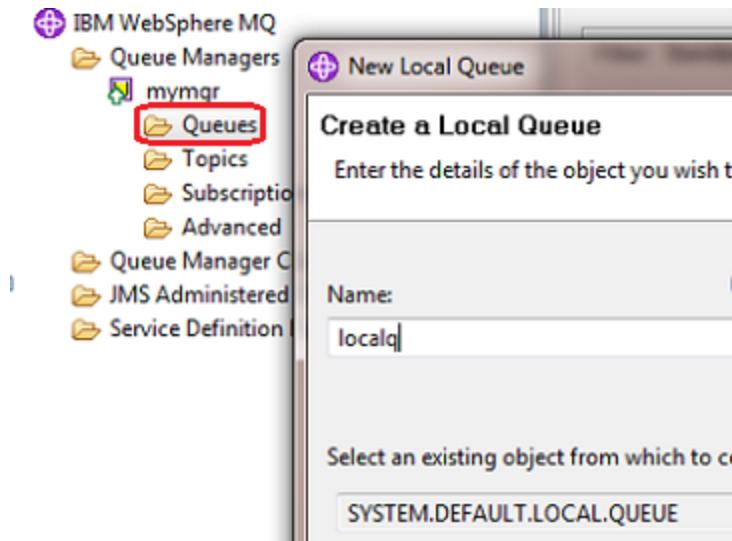
4. Restart the WebSphere MQ service.

Creating the Queue in WebSphere MQ

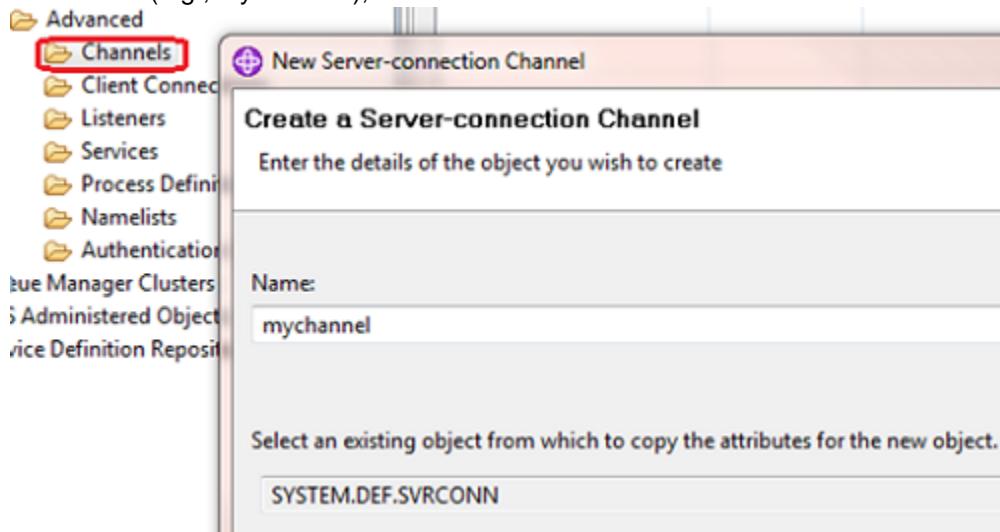
1. Start IBM WebSphere MQ Explorer and create a new queue manager. Make sure you select **make this the default queue manager** check box. Leave default values on the other fields.



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:

```
ALTER TCF(ivtTCF) TRANSPORT(CLIENT)
ALTER TCF(ivtTCF) QMGR(mymgr)
```

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.mqjms.jar
 - fscontext.jar
 - providerutil.jar
 - com.ibm.mq.jmqi.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.jmqi.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:

```

<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</pa
rameter>
                        <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"
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">queue</parameter>
                                </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 [Creating 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

Enter Event Publisher Details

Event Publisher Name*	IBMWebSphereOutputEventAdapter ② Enter a unique name to identify Event Publisher
From	Test Stream:1.0.0 ② The stream of events that need to be published
Event Source*	sensor id int
Stream Attributes	
To	jms ② Select the type of Adapter to publish events
Static Adapter Properties	
JNDI Initial Context Factory Class*	com.sun.jndi.fscontext.RefFSContextFactory ② JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.
JNDI Provider URL*	file:/C:/JNDI-Directory ② URL of the JNDI provider.
Username	jms-user
Password	*****
Connection Factory JNDI Name*	vtQCF ② The JNDI name of the connection factory.
Destination Type*	topic ② Type of the destination.
Destination*	test_topic
Dynamic Adapter Properties	
Header	header_name1:header_value1,header_name2:header_value2 ② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)
Mapping Configuration	
Message Format*	text ② Select the output message format
<input checked="" type="checkbox"/> Advanced	
<input type="button" value="Add 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.

```

<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</propert
y>
        <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">JP4yDiEh6HogOEjJzQQwHaJFIWZlnJTzaERl4eYrwukNeypm36R+odMk
aN9b2q4H9jBQsRV+mhcT1wQVnBpEZn4a+SuFuLKh3NihDEgww6R1tZVo8p1D6TUKvSHXYEpwSOgKrkOmdaFEOQ
OjfdhfK3Hrnjkz/MYPYQknrLK5MIY=</property>
        <property name="transport.jms.DestinationType">topic</property>
        <property
name="transport.jms.Header">header_name1:header_value1,header_name2:header_value2</pro
perty>
        <property name="transport.jms.Destination">test_topic</property>
        <property name="transport.jms.ConnectionFactoryJNDIName">ivtQCF</property>
    </to>
</eventPublisher>

```

Static adapter properties

Adapter Property	Description	Configuration file property
JNDI Initial Context Factory Class	The JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface	<code>java.naming.factory.initial</code>
JNDI Provider URL	URL of the JNDI provider	<code>java.naming.provider.url</code>
Username	Valid username for the JMS connection	<code>transport.jms.UserName</code>
Password	Valid password for the JMS connection	<code>transport.jms.Password</code>
Connection Factory JNDI Name	The JNDI name of the connection factory	<code>transport.jms.ConnectionFactoryJNDIN</code>
Destination Type	The sort order for messages that arrive on a specific destination	<code>transport.jms.DestinationType</code>
Destination	The topic or queue to which WSO2 CEP/DAS sends messages by publishing	<code>transport.jms.Destination</code>

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_val
---------------	-------------------------------------------------------------------------------	----------------------	----------------------------------------------------

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. Add the following Qpid JMS-specific JAR files to <PRODUCT_HOME>/repository/components/lib/ directory.
 - <ACTIVEMQ_HOME>/lib/geronimo-jms_1.1_spec-1.0.jar
 - <QPID-CLIENT_HOME>/lib/ qpid-client-xxx.jar
 - <QPID-CLIENT_HOME>/lib/ qpid-common-xxx.jar
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 [Creating 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

Enter Event Publisher Details

Event Publisher Name*	QpidJMSOutputEventAdapter ⓘ Enter a unique name to identify Event Publisher
From	Test Stream:1.0.0 ⓘ The stream of events that need to be published
Event Source*	sensor id int
Stream Attributes	
To	jms ⓘ Select the type of Adapter to publish events
Static Adapter Properties	
JNDI Initial Context Factory Class*	org.apache.qpid.jndi.PropertiesFileInitialContextFactory ⓘ JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.
JNDI Provider URL*	repository/conf/jndi.properties ⓘ 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*	test_topic
Dynamic Adapter Properties	
Header	header_name1:header_value1,header_name2:header_value2 ⓘ Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)
Mapping Configuration	
Message Format*	text ⓘ Select the output message format
+ Advanced	
<input type="button" value="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="QpidJMSOutputEventAdapter" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventpublisher">
    .....
    <to eventAdapterType="jms">
        <property
name="java.naming.factory.initial">org.apache.qpid.jndi.PropertiesFileInitialContextFa
ctory</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">JP4yDiEh6HogOEjJzQQwHaJF1WZlnJTzaER14eYrwukNeypm36R+odMk
an9b2q4H9jBQsRV+mhcT1wQVnBpEZn4a+SuFuLKh3NihDEgww6R1tZVo8p1D6TUKvSHXYEpwSOgKrkOmdaFEOQ
0jfdhfK3Hrnjkz/MYPYQknrLK5MIY=</property>
        <property name="transport.jms.DestinationType">topic</property>
        <property
name="transport.jms.Header">header_name1:header_value1,header_name2:header_value2</pro
perty>
        <property name="transport.jms.Destination">test_topic</property>
        <property
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    </to>
</eventPublisher>

```

The above adapter properties are described below.

Static adapter properties

Adapter Property	Description	Configuration file property
JNDI Initial Context Factory Class	The JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface	<code>java.naming.factory.initial</code>
JNDI Provider URL	URL of the JNDI provider	<code>java.naming.provider.url</code>
Username	Valid username for the JMS connection	<code>transport.jms.UserName</code>
Password	Valid password for the JMS connection	<code>transport.jms.Password</code>
Connection Factory JNDI Name	The JNDI name of the connection factory	<code>transport.jms.ConnectionFactoryJNDIN</code>
Destination Type	The sort order for messages that arrive on a specific destination	<code>transport.jms.DestinationType</code>
Destination	The topic or queue to which WSO2 CEP/DAS sends messages by publishing.	<code>transport.jms.Destination</code>

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

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** formats 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. Add the following JMS -specific JAR files to `<PRODUCT_HOME>/repository/components/lib/` directory
 - `<MB_HOME>/client-lib/andes-client-xxx.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-xxx.jar`
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.

Creating a WSO2 MB JMS event publisher

For instructions on creating a WSO2 MB JMS event publisher, see [Creating 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.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name*	WSO2MBJMSOutputEventAdapter ② Enter a unique name to identify Event Publisher
From	Test Stream:1.0.0 ② The stream of events that need to be published
Event Source*	sensor id int
Stream Attributes	
To	jms ② Select the type of Adapter to publish events
Static Adapter Properties	
JNDI Initial Context Factory Class*	org.wso2.andes.jndi.PropertiesFileInitialContextFactory ② JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.
JNDI Provider URL*	repository/conf/jndi.properties ② URL of the JNDI provider.
Username	jms-user
Password	*****
Connection Factory JNDI Name*	TopicConnectionFactory ② The JNDI name of the connection factory.
Destination Type*	topic ② Type of the destination.
Destination*	test_topic
Dynamic Adapter Properties	
Header	header_name1:header_value1,header_name2:header_value2 ② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)
Mapping Configuration	
Message Format*	text ② Select the output message format
<input checked="" type="checkbox"/> Advanced	
<input type="button" value="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="WSO2MBJMSOutputEventAdapter" statistics="disable"
trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    .....
    <to eventAdapterType="jms">
        <property
name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFileInitialContextFac
tory</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">JP4yDiEh6HogOEjJzQQwHaJF1WZlnJTzaER14eYrwukNeypm36R+odMk
an9b2q4H9jBQsRV+mhcT1wQVnBpEZn4a+SuFuLKh3NihDEgww6R1tZVo8p1D6TUKvSHXYEpwSOgKrkOmdaFEOQ
0jfdhfK3Hrnjkz/MYPYQknrLK5MIY=</property>
        <property name="transport.jms.DestinationType">topic</property>
        <property
name="transport.jms.Header">header_name1:header_value1,header_name2:header_value2</pro
perty>
        <property name="transport.jms.Destination">test_topic</property>
        <property
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    </to>
</eventPublisher>

```

The above adapter properties are described below.

Static adapter properties

Adapter Property	Description	Configuration file property
JNDI Initial Context Factory Class	The JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface	<code>java.naming.factory.initial</code>
JNDI Provider URL	URL of the JNDI provider	<code>java.naming.provider.url</code>
Username	Valid username for the JMS connection	<code>transport.jms.UserName</code>
Password	Valid password for the JMS connection	<code>transport.jms.Password</code>
Connection Factory JNDI Name	The JNDI name of the connection factory	<code>transport.jms.ConnectionFactoryJNDIN</code>
Destination Type	The sort order for messages that arrive on a specific destination	<code>transport.jms.DestinationType</code>
Destination	The topic or queue to which WSO2 CEP sends messages by publishing.	<code>transport.jms.Destination</code>

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_val

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. Copy the following client JAR files from <KAFKA_HOME>/lib/ directory to <PRODUCT_HOME>/repository/components/lib/ directory.
 - kafka_2.10-0.8.1.jar
 - zkclient-0.3.jar
 - scala-library-2.10.1.jar
 - zookeeper-3.3.4.jar
 - kafka-clients-0.8.2.1
 - metrics-core-2.2.0

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.

Creating a Kafka event publisher

For instructions on creating a Kafka event publisher, see [Creating Alerts](#).

Configuring global properties

The following global properties can be set for the Kafka event publisher type in the <DAS_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

Enter Event Publisher Details

Event Publisher Name* Enter a unique name to identify Event Publisher

From

Event Source* The stream of events that need to be published

Stream Attributes
`sensor id int`

To

Output Event Adapter Type* Select the type of Adapter to publish events

Static Adapter Properties

Meta Broker List* This is for bootstrapping and the producer will only use it for getting metadata. (eg - {host1:port1,host2:port2}, and the list can be a subset of brokers)

Optional Configuration Properties Define optional configuration properties (eg - {property_name1:property_value1, property_name2:property_value2 .. })

Dynamic Adapter Properties

Topic*

Mapping Configuration

Message Format* Select the output message format

+ Advanced

Add Event Publisher

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="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

Adapter Property	Description	Configuration file property	Example
Meta Broker List	This is for bootstrapping and the producer will only use it for getting metadata. The list can be a subset of brokers.	meta.broker.list	{host1:port1,host2:port2}
Optional Configuration Properties	Define optional configuration properties	optional.configuration	{property_name1:property_value1, property_name2:property_value2}

Dynamic adapter properties

Adapter Property	Possible Values	Description	Configuration file property	Example
Topic	sensorStream	Name of the Kafka topic to which, input messages are published	topic	test_topic

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 [Creating 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

Enter Event Publisher Details

Event Publisher Name* <input type="text" value="LoggerOutputEventAdapter"/> <div style="font-size: small; margin-top: -5px;">⑦ Enter a unique name to identify Event Publisher</div>	From Event Source* <input type="text" value="Test Stream: 1.0.0"/> <div style="font-size: small; margin-top: -5px;">⑦ The stream of events that need to be published</div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">sensor id int</div>				
Stream Attributes <div style="border: 1px solid #ccc; height: 100px; margin-top: 10px;"></div>					
To Output Event Adapter Type* <input type="text" value="logger"/> <div style="font-size: small; margin-top: -5px;">⑦ Select the type of Adapter to publish events</div>					
Dynamic Adapter Properties <table border="0" style="width: 100%;"> <tr> <td style="width: 30%;">Unique Identifier</td> <td><input type="text" value="log_id"/></td> </tr> <tr> <td></td> <td><div style="font-size: small; margin-top: -5px;">⑦ To uniquely identify a log entry</div></td> </tr> </table>		Unique Identifier	<input type="text" value="log_id"/>		<div style="font-size: small; margin-top: -5px;">⑦ To uniquely identify a log entry</div>
Unique Identifier	<input type="text" value="log_id"/>				
	<div style="font-size: small; margin-top: -5px;">⑦ To uniquely identify a log entry</div>				
Mapping Configuration Message Format* <input type="text" value="text"/> <div style="font-size: small; margin-top: -5px;">⑦ Select the output message format</div>					
+ 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.

```

<eventPublisher name="LoggerOutputEventAdapter" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventpublisher">
.....
<to eventAdapterType="logger">
<property name="uniqueId">log_id</property>
</to>
</eventPublisher>

```

The above adapter properties are described below.

Adapter Property	Description	Configuration file property	Example
Unique Identifier	A string of characters to uniquely identify a log entry	uniqueId	log_id

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
- Creating a MQTT event publisher
- Related samples

Prerequisites

Follow the steps below before starting the MQTT event publisher configuration.

1. Download **MQTT client library** (`mqtt-client-0.4.0.jar`).
2. Add the file to `<PRODUCT_HOME>/repository/components/lib/` directory.

Creating a MQTT event publisher

For instructions on creating a MQTT event publisher, see [Creating Alerts](#).

Configuring global properties

The following global properties can be set for the MQTT event publisher type in the `<DAS_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.

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
connectionKeepAliveInterval	The time interval in milliseconds at which a check should be carried out to identify inactive threads.	Integer	60

Configuring adapter properties

Specify the **Static** and **Dynamic Adapter Properties**, when creating a MQTT event publisher using the management console as shown below.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name* MQTTOutputEventAdapter
Enter a unique name to identify Event Publisher

From

Event Source* Test Stream:1.0.0
The stream of events that need to be published

Stream Attributes

sensor id int

To

Output Event Adapter Type* mqtt
Select the type of Adapter to publish events

Static Adapter Properties

Client Id	<input type="text" value="test-client"/> <small>client identifier is used by the server to identify a client when it reconnects, It used for durable subscriptions or reliable delivery of messages is required.</small>
Broker Url*	<input type="text" value="tcp://localhost:1883"/> <small>MQTT broker url</small>
Username	<input type="text" value="mqtt-user"/> <small>Username of the broker (if required)</small>
Password	<input type="text" value="mqtt-password"/> <small>Password of the broker (if required)</small>
Clean Session	<input type="text" value="true"/> <small>Persist topic subscriptions and ack positions across client sessions</small>
Quality of Service*	<input type="text" value="1"/>

Dynamic Adapter Properties

Topic*	<input type="text" value="test_topic"/>
--------	-----------------------------------------

Mapping Configuration

Message Format*	<input type="text" value="xml"/> <small>Select the output message format</small>
-----------------	----------------------------------------------------------------------------------

[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="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

Adapter Property	Description	Configuration file property	Example
Client Id	Client identifier is used by the server to identify a client when it reconnects, It used for durable subscriptions or reliable delivery of messages is required	clientId	test-client
Broker Url	MQTT broker URL. The same URL can be used for WSO2 MB when offset=0	url	tcp://localhost:1883
Username	Username of the broker	username	mqtt-user
Password	Password of the broker	password	mqtt-password
Clean Session	Whether to persist topic subscriptions and acknowledge positions across client sessions	cleanSession	true/false
Quality of Service	Quality of service for delivering messages between clients and servers. There are three QoS levels in MQTT are as follows. <ul style="list-style-type: none"> At most once (0) At least once (1) Exactly once (2) 	qos	0,1,2

Dynamic adapter properties

Adapter Property	Description	Configuration file property	Example
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.

```
<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 [Creating Alerts](#).

Configuring global properties

You can change the queries used to perform the standard database operations by adding the customised queries in the <DAS_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.

Attribute/activity	Current query
string	VARCHAR(255)
double	DOUBLE
integer	INT
long	BIGINT
float	FLOAT
createTable	CREATE TABLE \$TABLE_NAME (\$COLUMN_TYPES)

insertDataToTable	INSERT INTO \$TABLE_NAME (\$COLUMNS) VALUES (\$VALUES)
isTableExist	SELECT * FROM \$TABLE_NAME limit 1
updateTableRow	UPDATE \$TABLE_NAME SET \$COLUMN_VALUES WHERE \$CONDITION
comma	,
questionMark	?
equal	=
and	AND
selectAllColumnsDataTypeInTable	SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '\$TABLE_NAME'
selectFromTable	SELECT \$COLUMNS FROM \$TABLE_NAME
oracle.string	varchar2(255)
oracle.long	CLOB
oracle.double	BINARY_DOUBLE
oracle.isTableExist	SELECT * FROM \$TABLE_NAME WHERE ROWNUM = 1
oracle.selectAllColumnsDataTypeInTable	SELECT COLUMN_NAME, DATA_TYPE FROM USER_TAB_COLS WHERE TABLE_NAME = '\$TABLE_NAME'
mssql.string	varchar2(255)
mssql.isTableExist	SELECT TOP 1 * FROM \$TABLE_NAME
mssql.selectAllColumnsDataTypeInTable	SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = '\$TABLE_NAME'
h2.integer	varchar2(255)
h2.long	REAL
h2.selectAllColumnsDataTypeInTable	SHOW COLUMNS FROM \$TABLE_NAME

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

Enter Event Publisher Details

Event Publisher Name*	RDBMSOutputEventAdapter ⑦ Enter a unique name to identify Event Publisher
From	
Event Source*	Test Stream:1.0.0 ⑦ The stream of events that need to be published
sensor id int	
Stream Attributes	
To	
Output Event Adapter Type*	rdbms ⑦ Select the type of Adapter to publish events
Static Adapter Properties	
Data Source Name*	WSO2_CARBON_DB
Table Name*	test-table
Execution Mode*	insert ⑦ Choose between inserts or updates
Composite key columns	key,group ⑦ Attributes used for uniqueness checks for updates. Use "comma" to separate if more than one attribute is selected.
Mapping Configuration	
Message Format*	map ⑦ Select the output message format
<input checked="" type="checkbox"/> Advanced	
<input type="button" value="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](#).

An RDBMS publisher does not identify the persisted attributes of an event stream. If you want only the persisted attributes in the connected event stream to be published, the persisted attributes can be defined as advanced properties. For more information about persisting attributes, see [Configuring Data Persistence](#).

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="RDBMSOutputEventAdapter" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventpublisher">
    .....
    <to eventAdapterType="rdbms">
        <property name="datasource.name">WSO2_CARBON_DB</property>
        <property name="table.name">sensordata</property>
        <property name="execution.mode">insert</property>
        <property name="update.keys">sensor-key,sensor-group</property>
    </to>
</eventPublisher>

```

The above adapter properties are described below.

Adapter Property	Description	Configuration file Property	Example
Data Source Name	Name of the datasource	datasource.name	WSO2_CARBON_DB
Table Name	Name of the table	table.name	sensordata
Execution Mode	Type of the execution mode.	execution.mode	insert/update or insert
Composite key columns	Attributes used for uniqueness checks for updates. Use commas to separate if you enter more than one attribute. It is required to enter one or more attributes as composite key columns if you select update-or-insert for the Execution Mode property.	update.keys	sensor-key,sensor-group

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
- Creating a SMS event publisher
- Other post configurations that use SMS event publisher
- Related samples

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 .

```
<axisconfigname="AxisJava2.0">
    ...
    <transportSender class="org.apache.axis2.transport.sms.SMSSender" name="sms">
        <parameter name="systemType"></parameter>
        <parameter name="systemId">das1</parameter>
        <parameter name="password">das123</parameter>
        <parameter name="host">localhost</parameter>
        <parameter name="port">2775</parameter>
        <parameter name="phoneNumber">DAS1</parameter>
    </transportSender>
    ...
</axisconfig>
```

2. Copy the following libraries to <PRODUCT_HOME>/repository/components/lib/ directory.

- axis2-transport-sms-1.0.0.jar
- jsmpp-2.1.0.jar

Creating a SMS event publisher

For instructions on creating a SMS event publisher, see [Creating 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* <input type="text" value="SMSOutputEventAdapter"/> <div style="font-size: small; color: #ccc;">SMSOutputEventAdapter</div> <div style="color: #ccc;">⑦ Enter a unique name to identify Event Publisher</div>	From Event Source* <input type="text" value="Test Stream:1.0.0"/> <div style="font-size: small; color: #ccc;">⑦ The stream of events that need to be published</div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">sensor id int</div>	
Stream Attributes <div style="border: 1px solid #ccc; height: 100px; margin-top: 10px;"></div>		
To Output Event Adapter Type* <input type="text" value="sms"/> <div style="font-size: small; color: #ccc;">⑦ Select the type of Adapter to publish events</div>		
Dynamic Adapter Properties <table border="0"> <tr> <td style="vertical-align: top;"> Phone No* <input type="text" value="0716453453"/> <div style="font-size: small; color: #ccc;">⑦ Phone No where SMS needs to be send (eg: [country-code][number])</div> </td> </tr> </table>		Phone No* <input type="text" value="0716453453"/> <div style="font-size: small; color: #ccc;">⑦ Phone No where SMS needs to be send (eg: [country-code][number])</div>
Phone No* <input type="text" value="0716453453"/> <div style="font-size: small; color: #ccc;">⑦ Phone No where SMS needs to be send (eg: [country-code][number])</div>		
Mapping Configuration Message Format* <input type="text" value="text"/> <div style="font-size: small; color: #ccc;">⑦ Select the output message format</div> <div style="margin-top: 10px;"> + Advanced </div>		
<input type="button" value="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.

Adapter Property	Description	Configuration file property	Example
Phone No	Phone number of the SMS receiver in the following format: [country-code][number]	sms.no	0716453453

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=das1
password=das123
timeout=unlimited
```

3. Start SMSC Simulator by executing the following command:

java -cp smpp.jar:smssim.jar com.logica.smssim.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.

```
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
- 0 exit
> 1
Enter port number> 2775
Starting listener... started.
```

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 [Creating Alerts](#).

Configuring global properties

The following global properties can be set for the SOAP event publisher type in the `<DAS_HOME>/repository/conf/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.

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
axis2ClientConnectionTimeout	The number of milliseconds allowed to elapse before the Axis2 client connection times out.	Integer	10000
reuseHTTPClient	If this property is set to <code>true</code> , it is allowed to reuse the connection to the HTTP client for subsequent requests.	Boolean	<code>true</code>
autoReleaseConnection	If this property is set to <code>true</code> , inactive connections are automatically killed.	Boolean	<code>true</code>
maxConnectionsPerHost	The maximum number of connections allowed per host configuration.	Integer	50

Configuring adapter properties

Specify the **Dynamic Adapter Properties**, when creating a SOAP event publisher using the management console as shown below.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name* <input type="text" value="SOAPOutputEventAdapter"/> <div style="font-size: small; margin-top: -5px;">⑦ Enter a unique name to identify Event Publisher</div>	From Event Source* <input type="text" value="Test Stream:1.0.0"/> <div style="font-size: small; margin-top: -5px;">⑦ The stream of events that need to be published</div> <div style="border: 1px solid #ccc; padding: 5px; height: 100px; vertical-align: top; width: 100%;">sensor id int</div>																	
Stream Attributes <div style="border: 1px solid #ccc; padding: 5px; height: 100px; width: 100%;"></div>																		
To Output Event Adapter Type* <input type="text" value="soap"/> <div style="font-size: small; margin-top: -5px;">⑦ Select the type of Adapter to publish events</div>																		
Dynamic Adapter Properties <div style="border: 2px solid red; padding: 5px; margin-top: 5px;"> <table border="0"> <tr> <td>Url*</td> <td><input type="text" value="http://localhost:9763/services/Axis2LogService/log"/></td> <td><div style="font-size: small; margin-top: -5px;">⑦ Url of the remote service</div></td> </tr> <tr> <td>User Name</td> <td colspan="2"><input type="text" value="soap-user"/></td> </tr> <tr> <td>Password</td> <td colspan="2"><input type="password" value="*****"/></td> </tr> <tr> <td>SOAP Headers</td> <td colspan="2"><input type="text" value="header1: value1, header2: value2"/></td> <td><div style="font-size: small; margin-top: -5px;">⑦ Specify necessary SOAP headers. (e.g. "header1: value1, header2: value2")</div></td> </tr> <tr> <td>HTTP Headers</td> <td colspan="2"><input type="text" value="header1: value1, header2: value2"/></td> <td><div style="font-size: small; margin-top: -5px;">⑦ Specify necessary HTTP headers. (e.g. "header1: value1, header2: value2")</div></td> </tr> </table> </div>		Url*	<input type="text" value="http://localhost:9763/services/Axis2LogService/log"/>	<div style="font-size: small; margin-top: -5px;">⑦ Url of the remote service</div>	User Name	<input type="text" value="soap-user"/>		Password	<input type="password" value="*****"/>		SOAP Headers	<input type="text" value="header1: value1, header2: value2"/>		<div style="font-size: small; margin-top: -5px;">⑦ Specify necessary SOAP headers. (e.g. "header1: value1, header2: value2")</div>	HTTP Headers	<input type="text" value="header1: value1, header2: value2"/>		<div style="font-size: small; margin-top: -5px;">⑦ Specify necessary HTTP headers. (e.g. "header1: value1, header2: value2")</div>
Url*	<input type="text" value="http://localhost:9763/services/Axis2LogService/log"/>	<div style="font-size: small; margin-top: -5px;">⑦ Url of the remote service</div>																
User Name	<input type="text" value="soap-user"/>																	
Password	<input type="password" value="*****"/>																	
SOAP Headers	<input type="text" value="header1: value1, header2: value2"/>		<div style="font-size: small; margin-top: -5px;">⑦ Specify necessary SOAP headers. (e.g. "header1: value1, header2: value2")</div>															
HTTP Headers	<input type="text" value="header1: value1, header2: value2"/>		<div style="font-size: small; margin-top: -5px;">⑦ Specify necessary HTTP headers. (e.g. "header1: value1, header2: value2")</div>															
Mapping Configuration Message Format* <input type="text" value="xml"/> <div style="font-size: small; margin-top: -5px;">⑦ Select the output message format</div>																		
+ 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="SOAPOutputEventAdapter" statistics="disable" trace="disable"
xmlns="http://wso2.org/carbon/eventpublisher">
    .....
    <to eventAdapterType="soap">
        <property name="httpHeaders">header1: value1, header2: value2</property>
        <property name="username">soap-user</property>
        <property name="soapHeaders">header1: value1, header2: value2</property>
        <property encrypted="true"
name="password">h0vbApz3iQVMok/RyJn/AT51VMAGZHVMLLP2a3hkmBP+pKiKSNhUOUzVeHTPAe6Ko+gls6
ut1UAAdPP1ctWnZCU0Slw69FFJg7FJkLUzTgN2ZnyEMSRYbt/Kyq/WKJE08JeNptUaJYsEGhIkRpJg4ZVeOzXek
BJt3TxZ3C4H+06I=</property>
        <property name="url">http://localhost:9763/services/Axis2LogService/log</property>
    </to>
</eventPublisher>

```

The above adapter properties are described below.

Adapter Property	Description	Configuration file property	Example
Url	Destination web service URL	url	http://localhost:9763/services/Axis2LogService/log
User Name	Username token which is required to send event to a HTTPS endpoint.	username	soap-user
Password	Password token which is required to send event to a HTTPS endpoint.	password	soap-password
SOAP Headers	Necessary SOAP headers.	soapHeaders	header1: value1, header2: value2
HTTP Headers	Necessary HTTP headers.	httpHeaders	header1: value1, header2: value2"

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.

- [Creating an UI event publisher](#)
- [Related samples](#)

Creating an UI event publisher

For instructions on creating an UI event publisher, see [Creating Alerts](#).

Configuring global properties

The following global properties can be set for the UI event publisher type in the `<DAS_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.

Property Key	Description	Data Type	Default Value
eventQueueSize	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.	Integer	30
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

There are not any adapter-specific properties for the UI event publisher as shown below.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name* <input type="text" value="UIOutputEventAdapter"/> <div style="font-size: small; margin-top: -5px;">⑦ Enter a unique name to identify Event Publisher</div>	From Event Source* <input type="text" value="Test Stream:1.0.0"/> <div style="font-size: small; margin-top: -5px;">⑦ The stream of events that need to be published</div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">sensor id int</div>
Stream Attributes <div style="border: 1px solid #ccc; height: 100px; margin-top: 10px;"></div>	
To Output Event Adapter Type* <input type="text" value="ui"/> <div style="font-size: small; margin-top: -5px;">⑦ Select the type of Adapter to publish events</div>	
Mapping Configuration Message Format* <input type="text" value="wso2event"/> <div style="font-size: small; margin-top: -5px;">⑦ Select the output message format</div>	
Advanced	

[Add Event Publisher](#)

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 DAS publishes to a websocket server to which the DAS server needs to connect in order to publish events. However, if the events are to be published in a web socket client, that web socket client needs to connect to the inbuilt websocket server of the DAS. In such scenarios, use the [WebSocket Local Event Publisher](#).

- Prerequisites
- Creating a WebSocket event publisher
- Related samples

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 [Creating Alerts](#).

Configuring global properties

The following global properties can be set for the WebSocket event publisher type in the <DAS_HOME>/repository/conf/input-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.

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 Adapter Properties**, when creating a WebSocket event publisher using the management console as shown below.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name*	WebSocketOutputEventAdapter ⑦ Enter a unique name to identify Event Publisher
From	Test Stream:1.0.0 ⑦ The stream of events that need to be published
Event Source*	sensor id int
Stream Attributes	
To	websocket ⑦ Select the type of Adapter to publish events
Static Adapter Properties	
Websocket Server URL*	ws://localhost:9099 ⑦ URL of the web socket server which the events to be published; e.g. "ws://localhost:9099".
Mapping Configuration	
Message Format*	text ⑦ Select the output message format
Advanced	

[Add Event Publisher](#)

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** that you want to apply on the receiving 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 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	Description	Configuration file property	Example
Web Socket Server URL	URL of the WebSocket server to which you want to connect	websocket.server.url	ws://localhost:9099

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 DAS publishes to a websocket client. Websocket clients need to connect to the inbuilt websocket server of WSO2 DAS for the events to be published. However, when the DAS publishes to a web socket server, the DAS 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 [Creating Alerts](#).

Configuring global properties

The following global properties can be set for WebSocket local event publisher type in the <DAS_HOME>/repository/conf/output-event-adapters.xml file. These properties apply to all the publishers of the web socket-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.

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

There are not any adapter-specific properties for the WebSocket local event publisher as shown below.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name* <input type="text" value="WebSocketLocalOutputEventAdapter"/> <div style="font-size: small; margin-top: -5px;">⑦ Enter a unique name to identify Event Publisher</div>	From Event Source* <input type="text" value="Test Stream:1.0.0"/> <div style="font-size: small; margin-top: -5px;">⑦ The stream of events that need to be published</div> <div style="border: 1px solid #ccc; padding: 5px; height: 100px; vertical-align: top; width: 100%;">sensor id int</div>
Stream Attributes	
To Output Event Adapter Type* <input type="text" value="websocket-local"/> <div style="font-size: small; margin-top: -5px;">⑦ Select the type of Adapter to publish events</div>	
Mapping Configuration Message Format* <input type="text" value="text"/> <div style="font-size: small; margin-top: -5px;">⑦ Select the output message format</div>	
+ Advanced	

[Add Event Publisher](#)

When multi-tenancy is used, the URL formats used to publish events are as follows.

Super-tenant/Tenant	URL Formats
Super-tenant	<pre>ws://localhost:<DAS_Server_Port>/outputwebsocket/<publisher_name> wss://localhost:<DAS_SSL_Server_Port>/outputwebsocket/<publisher_name>;</pre>
Tenant	<pre>ws://localhost:<DAS_Server_Port>/t/<tenant_domain>/outputwebsocket/<publisher_name> wss://localhost:<DAS_SSL_Server_Port>/t/<tenant_domain>/outputwebsocket/<publisher_name>;</pre>

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.

Super-tenant/Tenant	URL Formats

Super-tenant	<code>ws://localhost:";9763"/outputwebsocket/WebSocketLocalOutputEventPublisher</code> <code>wss://localhost:";9443"/outputwebsocket/WebSocketLocalOutputEventPublisher";</code>
Tenant	<code>ws://localhost:";9763"/t/mycompany.com/outputwebsocket/WebSocketLocalOutputEventPublisher</code> <code>wss://localhost:";9443"/t/mycompany.com/outputwebsocket/WebSocketLocalOutputEventPublisher";</code>

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](#)

WSO2Event 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 [Creating Alerts](#).

Configuring adapter properties

Specify the **Static Adapter Properties**, when creating a WSO2Event event publisher using the management console as shown below.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name* <input type="text" value="WSO2EventOutputEventAdapter"/> <div style="color: #cc0000;">? Enter a unique name to identify Event Publisher</div>								
From								
Event Source* <input type="text" value="Test Stream:1.0.0"/> <div style="color: #cc0000;">? The stream of events that need to be published</div>	<pre>sensor id int</pre>							
Stream Attributes								
To								
Output Event Adapter Type* <input type="text" value="wso2event"/> <div style="color: #cc0000;">? Select the type of Adapter to publish events</div>	Static Adapter Properties <table border="0" style="width: 100%;"> <tr> <td style="width: 15%; vertical-align: top;"> Receiver URL* <input type="text" value="tcp://localhost:7661"/> <div style="color: #cc0000;">? Enter the Receiver Url</div> </td> </tr> <tr> <td style="width: 15%; vertical-align: top;"> Authenticator URL <input type="text" value="tcp://auth-host:7661"/> <div style="color: #cc0000;">? Enter the Authenticator Url</div> </td> </tr> <tr> <td style="width: 15%; vertical-align: top;"> User Name* <input type="text" value="wso2event-user"/> <div style="color: #cc0000;">? Enter the UserName</div> </td> </tr> <tr> <td style="width: 15%; vertical-align: top;"> Password* <input type="password" value="*****"/> <div style="color: #cc0000;">? Enter the Password</div> </td> </tr> <tr> <td style="width: 15%; vertical-align: top;"> Protocol* <input type="text" value="thrift"/> <div style="color: #cc0000;">? The communication protocol that will be used to published events</div> </td> </tr> <tr> <td style="width: 15%; vertical-align: top;"> Publishing Mode <input type="text" value="non-blocking"/> <div style="color: #cc0000;">? Select the how events should be published</div> </td> </tr> <tr> <td style="width: 15%; vertical-align: top;"> Publishing Timeout <input type="text" value="0"/> <div style="color: #cc0000;">? Timeout for the non-blocking Publishing Mode, default its '0' (fail immediately)</div> </td> </tr> </table>	Receiver URL* <input type="text" value="tcp://localhost:7661"/> <div style="color: #cc0000;">? Enter the Receiver Url</div>	Authenticator URL <input type="text" value="tcp://auth-host:7661"/> <div style="color: #cc0000;">? Enter the Authenticator Url</div>	User Name* <input type="text" value="wso2event-user"/> <div style="color: #cc0000;">? Enter the UserName</div>	Password* <input type="password" value="*****"/> <div style="color: #cc0000;">? Enter the Password</div>	Protocol* <input type="text" value="thrift"/> <div style="color: #cc0000;">? The communication protocol that will be used to published events</div>	Publishing Mode <input type="text" value="non-blocking"/> <div style="color: #cc0000;">? Select the how events should be published</div>	Publishing Timeout <input type="text" value="0"/> <div style="color: #cc0000;">? Timeout for the non-blocking Publishing Mode, default its '0' (fail immediately)</div>
Receiver URL* <input type="text" value="tcp://localhost:7661"/> <div style="color: #cc0000;">? Enter the Receiver Url</div>								
Authenticator URL <input type="text" value="tcp://auth-host:7661"/> <div style="color: #cc0000;">? Enter the Authenticator Url</div>								
User Name* <input type="text" value="wso2event-user"/> <div style="color: #cc0000;">? Enter the UserName</div>								
Password* <input type="password" value="*****"/> <div style="color: #cc0000;">? Enter the Password</div>								
Protocol* <input type="text" value="thrift"/> <div style="color: #cc0000;">? The communication protocol that will be used to published events</div>								
Publishing Mode <input type="text" value="non-blocking"/> <div style="color: #cc0000;">? Select the how events should be published</div>								
Publishing Timeout <input type="text" value="0"/> <div style="color: #cc0000;">? Timeout for the non-blocking Publishing Mode, default its '0' (fail immediately)</div>								
Mapping Configuration								
Message Format* <input type="text" value="wso2event"/> <div style="color: #cc0000;">? Select the output message format</div>	+ Advanced							

After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** that 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 output mapping types, see [Output 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.

```
<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/gYjdMpBaGloaCV/XgamNwSPsLglQ1ALTAlYBUTexgZ8JEizoz/WL9H
5Ncas1Dq/wMbVlL1OueUTXoL1Kcm63kEf1YWIkoD9ySk0FCFVFWgCsGhH8cAVabeCEEpE+qhq0bFoXTfqYTKjo
P2+F1B4EjhDsU7M=</property>
</to>
</eventPublisher>
```

The above adapter properties are described below.

Adapter Property	Description	Configuration file property	Example
Receiver URL	URL of the target receiver	receiverURL	tcp://localhost:7661
Authenticator URL	URL of the authenticator	authenticatorURL	tcp://auth-host:7661
User Name	Username for the listener	username	wso2event-user
Password	Password for the listener	password	wso2event-password
Protocol	The communication protocol that will be used to publish events	protocol	thrift/binary
Publishing Mode	Events publishing mode. Non-blocking refers to asynchronous publishing, and blocking refers to synchronous publishing	publishingMode	non-blocking/blocking
Publishing Timeout	Positive integer to denote the timeout for the non-blocking publishing mode	publishTimeout	0

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
- Sample 0058 - Publishing Custom WSO2 Events via WSO2Event Transport

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
- Implementing the `OutputEventAdapterFactory` class
- Exposing the custom event publisher as an OSGI service
- Deploying the custom event publisher

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 OutputEventAdapterException`

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.

```
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.ADAPTER_KEEP_ALIVE_TIME_NAME) != null) {
            defaultKeepAliveTime = Integer.parseInt(globalProperties.get(
                EmailEventAdapterConstants.ADAPTER_KEEP_ALIVE_TIME_NAME));
        } else {
            defaultKeepAliveTime =
EmailEventAdapterConstants.DEFAULT_KEEP_ALIVE_TIME_IN_MILLS;
        }
        if
(globalProperties.get(EmailEventAdapterConstants.ADAPTER_EXECUTOR_JOB_QUEUE_SIZE_NAME) != null) {
            jobQueSize = Integer.parseInt(globalProperties.get(
EmailEventAdapterConstants.ADAPTER_EXECUTOR_JOB_QUEUE_SIZE_NAME));
        } else {
            jobQueSize =
EmailEventAdapterConstants.ADAPTER_EXECUTOR_JOB_QUEUE_SIZE;
        }
        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(AdapterConstants.ADAPTER_MESSAGE_EMAIL SUBJECT);
        String[] emailIds =
dynamicProperties.get(AdapterConstants.ADAPTER_MESSAGE_EMAIL_ADDRESS)
        .replaceAll(" ",
"").split(AdapterConstants.EMAIL_SEPARATOR);
        String emailType =
dynamicProperties.get(AdapterConstants.APAPTER_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() {

```

```

        return false;
    }
}

```

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.

```

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));
    // set email subject
    Property subject = new
Property>EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_SUBJECT);
    subject.setDisplayName(
        resourceBundle.getString>EmailEventAdapterConstants.ADAPTER_MESSAGE_EMAIL_SUBJECT));
    subject.setRequired(true);
    //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(emailAddress);
    dynamicPropertyList.add(subject);
    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, globalProperties);
    }
}

```

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 `OutputEventAdaptorFactory` type. The following is a sample implementation of a service class for a custom defined publisher.

```

/**
 * @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 {
            OutputEventAdaptorFactory emailEventAdaptorFactory = new
EmailEventAdaptorFactory();

            context.getBundleContext().registerService(OutputEventAdaptorFactory.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 `internel\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 `<Das_Home>/repository/compo`

nents/dropins directory.

When you start the DAS 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.

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.

Mapping Configuration

Message Format* wso2event ③ Select the output message format

Advanced

WSO2Event Mapping

Output Event Stream *

Output Event Version *

Meta Data

Name	Value Of	Actions
id	meta_sensorId	Delete
name	meta_sensorName	Delete

Name : Value Of : meta_sensorName Add

Correlation Data

No Correlation Data properties Defined

Name : Value Of : meta_timestamp Add

Payload Data

Name	Value Of	Actions
humidity	humidity	Delete
value	sensorValue	Delete

Name : Value Of : sensorValue Add

The configuration XML file of the above sample mapping is as follows.

```

<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
    <from ... />
    <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>
    <to ... />
</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 - DAS 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.

Mapping Configuration

Message Format* xml Select the output message format

Advanced

XML Mapping

OutputMapping Content* In-Line Pick from registry

```
<sensorData>
<id>{{meta_sensorId}}</id>
<sensorValue>{{sensorValue}}</sensorValue>
<humidity>{{humidity}}</humidity>
</sensorData>
```

The configuration XML file of the above sample mapping is as follows.

```
<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 - DAS will convert this to an empty value and send out
 - Empty attribute - DAS 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 events of the server in the WSO2Event format to any JSON 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.

Mapping Configuration

The screenshot shows the 'Mapping Configuration' section of the WSO2 Data Analytics Server. At the top, there is a dropdown menu set to 'json' with a tooltip 'Select the output message format'. Below it is an 'Advanced' button. The main area is titled 'JSON Mapping' and contains a code editor for 'OutputMapping Content'. The code editor has two tabs: 'In-Line' (selected) and 'Pick from registry'. The content of the code editor is a JSON object:

```
{"sensorData": { "id": {{meta_sensorId}}, "value": {{sensorValue}}, "humidity": {{humidity}}, "correlation": {"long": {{correlation_longitude}}, "lat": {{correlation_latitude}}}}}
```

The configuration XML file of the above sample mapping is as follows.

```
<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 - DAS will send this as a null value
 - Empty attribute - DAS will send this as it is as an empty value
- **String Attributes**
 - null - It will be sent out as a null value
 - Empty Attribute - It will be published as an empty string attribute

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.

Mapping Configuration

The screenshot shows the 'Text Mapping' configuration page. At the top, there is a dropdown menu labeled 'text' with a tooltip 'Select the output message format'. Below this, there is an 'OutputMapping Content*' field containing sensor data mapping. The 'In-Line' radio button is selected. The content of the mapping is as follows:

```

Sensor Data
Sensor ID : {{meta_sensorId}}
Sensor Name : {{meta_sensorName}}
Sensor located at ({{correlation_longitude}}, {{correlation_latitude}})
Value : {{sensorValue}}
Humidity : {{humidity}}

```

The configuration XML file of the above sample mapping is as follows.

```

<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.

- **Non-String Attributes** (e.g. int, double, long etc.)
 - null - DAS will convert this to an empty value and send out
 - Empty attribute - DAS 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

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

Message Format* map ? Select the output message format

+ Advanced

Map Mapping		
Name	Value Of	Actions
id	meta_sensorId	Delete
name	meta_sensorName	Delete
value	sensorValue	Delete

Name : Value Of : sensorValue Add

The configuration XML file of the above sample mapping is as follows.

```
<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>
```

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 - DAS will send this as a null value
 - Empty attribute - DAS will send this as it is as an empty value
- **String Attributes**
 - null - It will be sent out as a null value
 - Empty Attribute - It will be published as an empty string attribute

Using Registry Resources for Output Mapping

WSO2 DAS 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 DAS 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.

- Save the following configuration as an XML file in a preferred location in your machine.

```
<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
<from ... />
<mapping customMapping="enable" type="text">
    <registry cacheTimeoutDuration="15">conf:/templates/en/message</registry>
</mapping>
<to ... />
</eventPublisher>
```

- Log into the WSO2 DAS Management Console.

- Define an event stream as follows. For detailed instructions to create event streams, see [Understanding Event Streams and Event Tables](#).

[Home](#) > [Manage](#) > [Event](#) > [Streams](#)

[Help](#)

Define New Event Stream

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	<input type="text" value="sensorsteam"/> <small>Name of the Event Stream</small>																
Event Stream Version*	<input type="text" value="1.0.0"/> <small>Version of the event stream (Eg : 1.0.0)</small>																
Event Stream Description	<input type="text"/>																
Event Stream Nick-Name	<input type="text"/> <small>Nick name of the event stream</small>																
Stream Attributes <table border="1"> <thead> <tr> <th colspan="2">Meta Data Attributes</th> <th>Actions</th> </tr> <tr> <th>Attribute Name</th> <th>Attribute Type</th> <th></th> </tr> </thead> <tbody> <tr> <td>sensorId</td> <td>int</td> <td> Delete</td> </tr> <tr> <td>sensorName</td> <td>string</td> <td> Delete</td> </tr> <tr> <td>language</td> <td>string</td> <td> Delete</td> </tr> </tbody> </table> Attribute Name : <input type="text"/> Attribute Type : <input type="button" value="string"/> Add			Meta Data Attributes		Actions	Attribute Name	Attribute Type		sensorId	int	Delete	sensorName	string	Delete	language	string	Delete
Meta Data Attributes		Actions															
Attribute Name	Attribute Type																
sensorId	int	Delete															
sensorName	string	Delete															
language	string	Delete															
Correlation Data Attributes <table border="1"> <thead> <tr> <th colspan="2">Attribute Name</th> <th>Attribute Type</th> <th>Actions</th> </tr> </thead> <tbody> <tr> <td>longitude</td> <td>double</td> <td> Delete</td> </tr> <tr> <td>latitude</td> <td>double</td> <td> Delete</td> </tr> </tbody> </table> Attribute Name : <input type="text"/> Attribute Type : <input type="button" value="double"/> Add			Attribute Name		Attribute Type	Actions	longitude	double	Delete	latitude	double	Delete					
Attribute Name		Attribute Type	Actions														
longitude	double	Delete															
latitude	double	Delete															
Payload Data Attributes <table border="1"> <thead> <tr> <th colspan="2">Attribute Name</th> <th>Attribute Type</th> <th>Actions</th> </tr> </thead> <tbody> <tr> <td>sensorValue</td> <td>double</td> <td> Delete</td> </tr> </tbody> </table> Attribute Name : <input type="text"/> Attribute Type : <input type="button" value="double"/> Add			Attribute Name		Attribute Type	Actions	sensorValue	double	Delete								
Attribute Name		Attribute Type	Actions														
sensorValue	double	Delete															
Add Event Stream																	

- Enter basic information for the stream as follows.

Parameter	Value
Event Stream Name	sensorsteam
Event Stream Version	1.0.0

Add attributes as follows.

Attribute Category	Attribute Name	Attribute Type
Meta Data	sensorId	int
	sensorName	string
	language	string
Correlation Data	longitude	double
	latitude	double
Payload Data	sensorValue	double

- Add the XML file you created as a registry resource as follows. For more information about the registry, see [Registry](#).

- In the **Main** tab expand the **Registry** section and click **Browse**.
- 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.
- Navigate to the /_system/config/Template sub directory and click **Add Resource**. Then enter information as shown in the table below, and click **Add**.

Field	Value
Method	Upload content from file
File	Browse and select the XML file you saved in step 1.
Name	CustomOutputMapping
Media Type	application/xml

- Create a new publisher as follows. For more information, see [Creating Alerts - Creating event publishers](#).

Field	Value
Event Publisher Name	samplelogger
Event Source	sensorstream
Output Event Adapter Type	logger

- 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.

Text Mapping

OutputMapping Content*	<input type="radio"/> In-Line <input checked="" type="radio"/> Pick from registry
Registry Path*	<input type="text" value="conf:/Template/OutputMapping.xml"/>
Cache Timeout (in minutes)*	<input type="text" value="15"/> <small>15 Enter a positive cache timeout period in minutes. For immediate timeout, use 0.</small>

In this example, the default value of 15 is left unchanged for the **Cache Timeout (in minutes)** field. Therefore, this resource will be cached every 15 minutes.

7. Click **Add Event Publisher**.
8. 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.

Parameter	Value
sensorid	10
sensorName	Temperature
language	en
longitude	79.861256
latitude	6.927131
sensorValue	23.0

The output for this event is logged in the terminal as follows.

```

Sri L
[2016-08-15 12:24:11,267] INFO {org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to the junction. Stream:sensorstream:1.0.0
[2016-08-15 12:24:11,270] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: samplelogger,
Event: Welcome
Sensor Temperature reports 23.0 from 79.861256:6.927131 ts.csv
File Stream Configuration Delay between events(ms) Action
click the configure button click the configure button
Browse... No file selected upload

```

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.

The sensor stream event stream used in the previous tutorial is also used in this tutorial.

1. Log into the DAS 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](#).

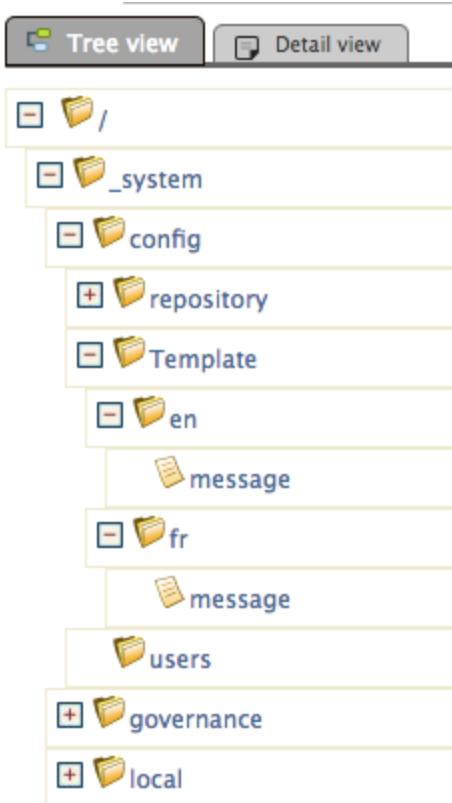
- In the **Main** tab expand the **Registry** section and click **Browse**.
- Under **Tree View**, navigate to the `/_system/config/Template` sub directory. Create two more sub directories in it named `en` and `fr`.
- Navigate to the `/_system/config/Template/en` sub directory and click **Add Resource**. Then enter information as shown in the table below, and click **Add**.

Field	Value
Method	Create Text content
Name	message
Media Type	text/plain
Content	<pre>Welcome to sensor data Sensor ID : {{meta_sensorId}} Sensor Name : {{meta_sensorName}} Sensor located at ({{correlation_longitude}}, {{correlation_latitude}}) Value : {{sensorValue}} Humidity : {{humidity}}</pre>

- 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**.

Field	Value
Method	Create Text content
Name	message
Media Type	text/plain
Content	<pre>Bienvenue to capteur données Sensor ID : {{meta_sensorId}} Sensor Name : {{meta_sensorName}} Sensor located at ({{correlation_longitude}}, {{correlation_latitude}}) Value : {{sensorValue}} Humidity : {{humidity}}</pre>

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 [Creating Alerts - Creating event publishers](#).

Field	Value
Event Publisher Name	samplelogger
Event Source	sensorstream
Output Event Adapter Type	logger

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.

Field	Value
Registry Path	conf:/templates/{meta_language}/message
Cache Timeout (in minutes)	15

5. Log into the DAS 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.

Parameter	Value for Event 1	Value for Event 2	Value for Event 3
sensorid	10	10	10
sensorName	Temperature	Temperature	Temperature
language	en	fr	ru

longitude	79.861256	79.861256	79.861256
latitude	6.927131	6.927131	6.927131
sensorValue	23.0	23.0	23.0

The three events are logged as follows.

```
[2016-08-15 12:45:51,373] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: samplelogger,
Event: Welcome
Sensor Temerature reports 23.0 from 79.861256:6.927131
Stream Attributes
[2016-08-15 12:46:05,214] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: samplelogger,
Event: Bienvenue
Sensor Temerature reports 23.0 from 79.861256:6.927131
[2016-08-15 12:46:17,562] ERROR {org.wso2.carbon.event.publisher.core.internal.EventPublisher} - Cannot send
Event{
    streamId='sensorstream:1.0.0', language(string) *
    timeStamp=147124537560,
    metaData=[10, Temerature, ta], Correlation Attributes
    correlationData=[79.861256, 6.927131], payloadType(double) *
    payloadData=[23.0],
    arbitraryDataMap=null,
}
from samplelogger
Payload Attributes
org.wso2.carbon.event.publisher.core.exception.EventPublisherConfigurationException: Error in getting cached resource
    at org.wso2.carbon.event.publisher.core.internal.util.RuntimeResourceLoader.getResourceContent(RuntimeResourceLoader.java:59)
    at org.wso2.carbon.event.publisher.core.internal.type.text.TextOutputMapper.convertToMappedInputEvent(TextOutputMapper.java:126)
```

Note that an error is returned for the third event. This is because no output mapping was done for the `ru` language.

Communicating Results Through REST API

WSO2 DAS stores the received events and processed data in its underlying data storage system, where that data can be retrieved using the standards APIs that have been defined. [Analytics REST API](#) allows external parties to query this data. When this API is used independently of the backend data store, it can be used to lookup/search for data that is stored in the system. e.g., this can be used by any server side application, mobile application etc. anywhere the HTTP based service can be accessed.

Analytics JavaScript (JS) API

Analytics JavaScript API exposes WSO2 DAS analytics functionalities as JavaScript functions. You can use this JavaScript API in your Web apps to perform analytics operations when creating Web apps or Dashboard gadgets. You can find the JavaScript API in the `<DAS_HOME>/repository/deployment/server/jaggeryapps/portal/js/carbon-analytics.js` file. This can be exposed as follows: `<DAS_URL>:<DAS_PORT>/portal/js/carbon-analytics.js`

- Creating the analytics client for the JS API
- Success callback function and error callback functions of the JS API

Creating the analytics client for the JS API

You need to first create a client to use the JavaScript API. Create an instance of the `AnalyticsClient` module as follows.

1. Import `jquery.js` and `carbon-analytics.js`.
2. Create the connection object with the following details.

```
var username = "admin";
var password = "admin";
var server_url = "https://localhost:9443/portal/apis/analytics";
var client = new AnalyticsClient().init(username, password, server_url);
```

Currently due to a limitation, client applications should be deployed in the same domain as the Dashboard

server. i.e - DAS_HOME/repository/deployment/server/webapps

Success callback function and error callback functions of the JS API

All methods of the Analytics JS API have a success function and an error function as callbacks representing successful invocations and erroneous invocations. Success callback has one argument which will contain the response it returns if the invocation is successful. It represents a JSON String of the following format.

```
{
    "status" : <RESPONSE_STATUS>,
    "message" : <RESPONSE_DATA_OR_MESSAGE>
}
```

Error callback has one argument which will contain an error message if the invocation is failed. Its JSON representation is as follows.

```
{
    "status" : <RESPONSE_STATUS>,
    "message" : <ERROR_MESSAGE>
}
```

The methods exposed by the Analytics JavaScript API are as follows.

- Retrieving the List of Tables of All Record Stores via JS API
- Retrieving All Record Stores via JS API
- Retrieving the Record Store of a Given Table via JS API
- Checking if a Given Table Exists via JS API
- Clearing Indexed Data of a Given Table via JS API
- Retrieving Records Based on a Time Range via JS API
- Retrieving Records Matching the Given Primary Key Combination via JS API
- Retrieving Records of Given Record IDs via JS API
- Retrieving the Total Record Count of a Table via JS API
- Retrieving the Number of Records Matching the Given Search Query via JS API
- Retrieving All Records Matching the Given Search Query via JS API
- Retrieving the Schema of a Table via JS API
- Checking if the Given Records Store Supports Pagination via JS API
- Tracking the Indexing Process Completion via JS API
- Drilling Down Through Categories via JS API
- Retrieving Specific Records through a Drill Down Search via JS API
- Retrieving the Number of Records Matching the Drill Down Criteria via JS API
- Adding an Event Stream Definition via JS API
- Publishing Events to WSO2 DAS via JS API
- Retrieving an Existing Event Definition via JS API
- Tracking the Indexing Process Completion of a Table via JS API
- Retrieving Aggregated Values of Given Records via JS API

Retrieving the List of Tables of All Record Stores via JS API

- Overview
- Example
- Sample output

Overview

Function	listTables(success, error)
Description	Lists all tables of all the record stores.
Output	A JSON String array containing the table names in the 'message' element within the argument of the success callback.

Example

```
client.listTables(function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
[ "TABLE1", "TABLE2", "TABLE3" ]
```

Retrieving All Record Stores via JS API

- [Overview](#)
- [Example](#)
- [Sample output](#)

Overview

Function	listRecordStores(success, error)
Description	Lists all the record stores.
Output	A JSON String array containing the record store names in the 'message' element within the argument of the success callback.

Example

```
e.g.
client.listRecordStores(function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
[ "EVENT_STORE", "PROCESSED_DATA_STORE" ]
```

Retrieving the Record Store of a Given Table via JS API

- Overview
- Example
- Sample output

Overview

Function	getRecordStoreByTable(tableName, success, error)
Description	Returns the records store of the given table.
Output	The record store of the given table.

Example

```
client.getRecordStoreByTable("SampleTable", function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
EVENT_STORE
```

Checking if a Given Table Exists via JS API

- Overview
- Example
- Sample output

Overview

Function	tableExists(tableName, success, error)
Description	Checks if a given table exists.
Output	A JSON object containing the following text if the given table; <ul style="list-style-type: none"> • exists - "Table : testtable exists." • does not exist - "Table : testtable does not exist."

Example

```
client.tableExists("SampleTable", function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
{
status: "success"
message: "Table : testtable exists."
}
```

Clearing Indexed Data of a Given Table via JS API

- Overview
- Example
- Sample output

Overview

Function	clearIndexData (tableName, success, error)
Description	Deletes all indexed data of the given table.
Output	A JSON object containing the message in the 'message' element within the argument of the success callback.

Example

```
client.clearIndexData("SampleTable", function(data) {
    console.log (data[ "message" ]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
{
status: "success"
message: "Successfully cleared indices in table: SampleTable"
}
```

Retrieving Records Based on a Time Range via JS API

- Overview
- Example
- Sample output

Overview

Function	getRecordsByRange(rangeInfo, success, error)
Description	Returns all the records which fall between the given time range. Additionally, you can provide pagination information.
Output	A JSON String array containing the records in the 'message' element within the argument of the success callback.

Example

```

var rangeInfo = {
    tableName : "sampleTable", //name of the table
    timeFrom : 143435365300, //lower bound of the time range inclusive
    timeTo : 143435365343, //upper bound of time range exclusive
    start : 0, // starting index of records
    count : 10, //page size for pagination
    columns : [ "column1", "column2"] //interested columns if null, return all the
columns
};

client.getRecordsByRange(rangeInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

Sample output

```
[ {
    "id": "14399961246350.13125980939205978",
    "tableName": "sampleTable", "timestamp": 1439996124610,
    "values": {
        "column1": "234,454,34\u00027",
        "column2": 23
    }
}]
```

Retrieving Records Matching the Given Primary Key Combination via JS API

- Overview
- Example
- Sample output

Overview

Function	getWithKeyValues(recordInfo, success, error)
Description	Returns all the records in a table which match the given primary key value combinations.
Output	A JSON String array containing the table names in the 'message' element within the argument of the success callback.

Example

```

var recordInfo = {
    tableName : "TEST", // table being accessed
    valueBatches : [
        { //key value pairs of the first batch
            column1 : "value1",
            column2 : "value2"
        },
        { //key value pairs of the second batch
            column1 : "anotherValue1",
            column2 : "anotherValue2"
        }
    ],
    columns : [ "column1" ] //interested columns, if null, return all the columns
};

client.getWithKeyValues(recordInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

Sample output

```
[
  {
    "id": "14399961246350.13125980939205978",
    "tableName": "TEST", "timestamp": 1439996124610,
    "values": {
      "column1": "value1",
      "column2": "value2"
    }
  },
  {
    "id": "14399961246350.13125980939205999",
    "tableName": "TEST", "timestamp": 1439996124610,
    "values": {
      "column1": "anotherValue1",
      "column2": "anotherValue2"
    }
  }
]
```

Retrieving Records of Given Record IDs via JS API

- Overview
- Example
- Sample output

Overview

Function	getRecordsByIds(recordsInfo, success, error)
Description	Returns all the records which match the given record IDs and the table.

Output	A JSON String array containing the matching records in the 'message' element within the argument of the success callback.
---------------	---------------------------------------------------------------------------------------------------------------------------

Example

```
var recordsInfo = {
    tableName : "TEST", //table being accessed
    ids : [ "14399961246350.13125980939205978", "14399961246350.13125980939205999" ]
//interested ids
};

client.getRecordsByIds(recordsInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
[ {
    "id": "14399961246350.13125980939205978",
    "tableName": "TEST", "timestamp": 1439996124610,
    "values": {
        "column1": "value1",
        "column2": "value2"
    }
},
{
    "id": "14399961246350.13125980939205999",
    "tableName": "TEST", "timestamp": 1439996124610,
    "values": {
        "column1": "anotherValue1",
        "column2": "anotherValue2"
    }
}

]
```

Retrieving the Total Record Count of a Table via JS API

- [Overview](#)
- [Example](#)
- [Sample output](#)

Overview

Function	getRecordCount(tableName, success, error)
Description	Returns the total record count of a table.
Output	The record count in the 'message' element within the argument of the success callback.

Example

```
client.getRecordCount("SampleTable", function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occured: " + error);
});
```

Sample output

3

Retrieving the Number of Records Matching the Given Search Query via JS API

- Overview
- Example
- Sample output

Overview

Function	searchCount(queryInfo, success, error)
Description	Returns the number of records which matches the given search query.
Output	The number of records in the 'message' element within the argument of the success callback.

Example

```
var queryInfo = {
    tableName : "SampleTable",
    searchParams : {
        query : <ucene-query>
    }
};
client.searchCount(queryInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occured: " + error);
});
```

Sample output

3

Retrieving All Records Matching the Given Search Query via JS API

- Overview
- Example
- Sample output

Overview

Function	search (queryInfo, success, error)
Description	Returns all records which match the given search query and sort them by the given sortBy parameters. records can be sorted by multiple fields. If two records have the same value for a specific sorting field, it will consider the second field to sort.
Output	A JSON String array containing the records in the 'message' element within the argument of the success callback.

Example

```

var queryInfo = {
    tableName : "TEST", //table being queried
    searchParams : {
        query : "column1:value1", //lucene query to search the records
        start : 0, //starting index of the matching record set
        count : 100 //page size for pagination
        sortBy : [
            {
                field : "column1", //this column will be used to sort the records in
                DESC order
                sortType : "DESC"
            },
            {
                field : "column2", // if there are records with the same value for
                field "column1" then this field ( "column2" ) will be considered to sort only those
                records
                sortType : "ASC"
            }
        ]
    }
};

client.search(queryInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

Sample output

```
[ {
    "id": "14399961246350.13125980939205978",
    "tableName": "TEST", "timestamp": 1439996124610,
    "values": {
        "column1": "value1",
        "column2": "value2"
    }
},
{
    "id": "14399961246350.13125980939205999",
    "tableName": "TEST", "timestamp": 1439996124610,
    "values": {
        "column1": "value1",
        "column2": "anotherValue2"
    }
}
]
```

Retrieving the Schema of a Table via JS API

- [Overview](#)
- [Example](#)
- [Sample output](#)

Overview

Function	getSchema(tableName, success, error)
Description	Returns the schema of a table.
Output	A JSON object containing the schema of the table in the 'message' element within the argument of the success callback.

Example

```
client.getSchema("SampleTable", function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
{
  columns:
  {
    column1: {
      type: STRING,
      isScoreParam: FALSE,
      isIndex: TRUE
    },
    column2: {
      type: INTEGER,
      isScoreParam: TRUE,
      isIndex: TRUE
    }
  }
  primaryKeys: [column1, column2]
}
```

Checking if the Given Records Store Supports Pagination via JS API

- Overview
- Example
- Sample output

Overview

Function	isPaginationSupported(recordStore, success, error)
Description	Checks if the given records store supports pagination or not.
Output	A JSON object containing the following text in 'message' element if the given table; <ul style="list-style-type: none"> • if supports pagination - returns "true" • if does not support pagination - returns "false"

Example

```
client.isPaginationSupported("SampleRecordStore", function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
true
```

Tracking the Indexing Process Completion via JS API

- Overview
- Example
- Sample output

Overview

Function	waitForIndexing(success, error)
Description	Tracks if the indexing process is completed.
Output	A JSON String object containing the message on successful completion of the indexing process.

Example

```
client.waitForIndexing(function(data) {
    console.log (data[ "message" ]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
{
  status: "success"
  message: "Indexing completed successfully"
}
```

Drilling Down Through Categories via JS API

- [Overview](#)
- [Example](#)
- [Sample output](#)

Overview

Function	drillDownCategories(drilldownRequest, success, error)
Description	Returns the child categories of the given category along with their scores.
Output	A JSON object containing child categories along with their scores in the 'message' element within the argument of the success callback.

Example

```

var drillDownReq = {
    tableName : "TEST", //tableName
    drillDownInfo : {
        fieldName : "facetField1", //field which is indexed as a FACET
        categoryPath : [ "category", "subCategory"] //Path being drilled
    down, optional
        query : "logFile : wso2carbon.log" //search query, optional
        scoreFunction : "sqrt(weight)" //score function
    }
},
client.drillDownCategories(drillDownReq, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

If categoryPath is not provided, it takes the root element of the FACET field as the default path. Therefore, the top level facets are returned.

Sample output

```
{
    "categoryPath" : [category, subCategory],
    "categories" : [child1, child2, child3]
}
```

Retrieving Specific Records through a Drill Down Search via JS API

- [Overview](#)
- [Example](#)
- [Sample output](#)

Overview

Function	drillDownSearch(drilldownRequest, success, error)
Description	Returns records which match the given facet path/category path and other search options.
Output	A JSON String array containing the records in the 'message' element within the argument of the success callback.

Example

```

var drillDownReq = {
    tableName : "TEST", //table name
    drillDownInfo : {
        categories : [ // list of facet fields to match
            {
                fieldName : "facetField1", //Only records which has the path A,
B, C in facetField1
                path : ["Srilanka", "Colombo"]
            },
            {
                fieldName : "facetField2", //Only records which has the path X,
Y, Z in facetField2
                path : [ "2015", "Mar", "23"]
            }
        ],
        query : "field1 : value1", //Additional lucene query, optional
        recordStart : 0, //starting indexing of matching record set
        recordCount : 50, // paging size for pagination
        sortBy : [
            {
                field : "field1", //first field which is used to sort the records
                sortType : "DESC" //type of sorting
            },
            {
                field : "field2", //This field will be used to sort the records if field1 has
the same value for some records, so those record will be sorted by the second field
"field2"
                sortType : "ASC"
            }
        ]
    };
}

client.drillDownSearch(drillDownReq, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

Sample output

```
[
  {
    "id": "14242656601230.5739142271249453",
    "tableName": "TEST",
    "timestamp": 1424265660123,
    "values": {
      "product": "wso2cdm",
      "field1": 23,
      "field2": 43,
      "facetField1": ["Srilanka", "Colombo"],
      "facetField2": ["2015", "Mar", "23"]
    }
  },
  {
    "id": "14242656601230.5739142271249451",
    "tableName": "TEST",
    "timestamp": 1424265660123,
    "values": {
      "product": "wso2cdm",
      "field1": 23,
      "field2": 41,
      "facetField1": ["Srilanka", "Colombo"],
      "facetField2": ["2015", "Mar", "23"]
    }
  }
]
```

Retrieving the Number of Records Matching the Drill Down Criteria via JS API

- Overview
- Example
- Sample output

Overview

Function	drilldownSearchCount(drilldownRequest, success, error)
Description	Returns the number of records which match the given drill down criteria.
Output	The number of records in the 'message' element within the argument of the success callback.

Example

```

var drillDownReq = {
    tableName : "TEST", //table name
    drillDownInfo : {
        categories : [ // list of facet fields to match
            {
                fieldName : "facetField1", //Only records which has the path A,
                B, C in facetField1
                path : [ "A", "B", "C"]
            },
            {
                fieldName : "facetField2", //Only records which has the path X,
                Y, Z in facetField2
                path : [ "X", "Y", "Z"]
            }
        ],
        query : "field1 : value1", //Additional lucene query
        scoreFunction : "scoreParamField * 2" //score function
    };
}

client.drillDownSearchCount(drillDownReq, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

Sample output

3

Adding an Event Stream Definition via JS API

- Overview
- Example
- Sample output

Overview

Function	addStreamDefinition(StreamDefinition, success, error)
Description	Returns the added event stream definition.
Output	A JSON object containing the added event stream definition stream ID in the 'message' element within the argument of the success callback.

Example

```

var streamDef = {
    name : "TEST",
    version : "1.0.0",
    nickName : "test",
    description : "sample description"
    payloadData : {
        name : "STRING",
        married : "BOOLEAN",
        age : "INTEGER"
    },
    metaData : {
        NIC: "LONG"
    },
    correlationData : {
    },
    tags : []
};
client.addStreamDefinition(streamDef, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

Sample output

```
TEST:1.0.0
```

Publishing Events to WSO2 DAS via JS API

- [Overview](#)
- [Example](#)
- [Sample output](#)

Overview

Function	publishEvent(event, success, event)
Description	Returns the events published to WSO2 DAS.
Output	A JSON object containing the response text in the 'message' element within the argument of the success callback.

Example

```

var event = {
    streamName : "TEST",
    streamVersion : "1.0.0",
    payloadData : {
        name : "John",
        married : "false",
        age : "27"
    },
    metaData : {
        NIC: "12345678"
    },
    correlationData : {
    },
    arbitraryDataMap : {
    }
};
client.publishEvent(event, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

Sample output

"Event published successfully"

Retrieving an Existing Event Definition via JS API

- Overview
- Example
- Sample output

Overview

Function	getStreamDefinition(requestInfo, success, error)
Description	Returns the existing event definition which matches the given search criteria.
Output	A JSON object containing the event definition in the 'message' element within the argument of the success callback.

Example

```

var requestInfo = {
    name : "TEST",
    version" "1.0.0"
};
client.getStreamDefinition(requestInfo, function(data) {
    console.log (data[ "message" ]);
}, function(error) {
    console.log("error occurred: " + error);
});

```

Sample output

The resulting ‘message’ element of the response is similar to the JSON object you send to [create an event stream definition](#).

```

{
    name : "TEST",
    version : "1.0.0",
    nickName : "test",
    description : "sample description"
    payloadData : {
        name : "STRING",
        married : "BOOLEAN",
        age : "INTEGER"
    },
    metaData : {
        NIC: "LONG"
    },
    correlationData : {
    },
    tags : []
}

```

Tracking the Indexing Process Completion of a Table via JS API

- [Overview](#)
- [Example](#)
- [Sample output](#)

Overview

Function	waitForIndexing(tableName, success, error)
Description	Tracks if the indexing process is completed.
Output	A JSON String object containing the message on successful completion of the indexing process.

Example

```
client.waitForIndexing("testTable", function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error);
});
```

Sample output

```
{
  status: "success"
  message: "Indexing completed successfully"
}
```

Retrieving Aggregated Values of Given Records via JS API

- Overview
- Example
- Sample output

Overview

Function	searchWithAggregates (queryInfo, success, error)
Description	Returns the aggregated values of the given records.
Output	A JSON String object containing the aggregated values of the given records, in the 'message' element within the argument of the success callback.

Example

```

var queryInfo = {
    tableName:"TEST",
    searchParams : {
        groupByField:"single_valued_facet_field",
        query : <lucene-query>
        fields:[
        {
            fieldName:"n",
            aggregate:"AVG",
            alias:result_avg
        },
        {
            fieldName:"n",
            aggregate:"MAX",
            alias:"result_max"
        }
    ]
},
client.searchWithAggregates(queryInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occured: " + error);
})�;

```

Sample output

```

[ {
    "id": "14399961246350.13125980939205978",
    "tableName": "TEST", "timestamp": 1439996124610,
    "values": {
        "result_avg": "20",
        "result_max": "22",
    "single_valued_facet_field": [engineering] //This should be a facet with a single
value.
    }
},
{
    "id": "14399961246350.13125980939205999",
    "tableName": "TEST", "timestamp": 1439996124610,
    "values": {
        "result_avg": "34",
        "result_max": "46"
    "single_valued_facet_field": [marketing] //This should be a facet with a single
value.
    }
}
]

```

Packaging Artifacts as a C-App Archive

C-App namely Carbon Application is an archive format collection of different artifacts bundled to a single deployable component. C-App files have CAR extensions and can be deployed to different runtimes. Each runtime will only

deploy the artifacts which match with the role that the runtime is playing.

- Supported C-App types
- Creating a C-App
- Deploying a C-App

Supported C-App types

Given below are the type of the carbon applications artifacts that are supported by WSO2 DAS.

Generic artifacts

Artifact	Type
Event Streams	event/streams
Event Receivers	event/receiver
Event Publishers	event/publisher

Real time analytics specific artifacts

Artifact	Type
Execution Plan	event/execution-plan

Batch analytics specific artifacts

Artifact	Type
Event stores	analytics/eventstore
Spark scripts	analytics/spark
Data purging scripts	analytics/dataPurging

Visualization artifacts

Artifact	Type
Dashboards	dashboards/dashboard
Layouts	dashboards/layout
Gadgets	dashboards/gadget

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 to be deployed in WSO2 DAS. In this below section, we are going to focus on adding the carbon application with 5 different kind of artifact event streams, event receiver, event publisher, event store, and Spark scripts.

```

artifacts.xml
Eventreceiver_1.0.0
    artifact.xml
    TestWso2EventReceiver.xml
EventPublisher_1.0.0
    artifact.xml
    TestLoggerPublisher.xml
Eventstore_1.0.0
    artifact.xml
    ORG_WSO2_TEST.xml
Eventstream_1.0.0
    artifact.xml
    org.wso2.test_1.0.0.json
Sparkscripts_1.0.0
|   artifact.xml
|   sample_script.xml
DataPurging_1.0.0
    artifact.xml
    data_purging.xml
Dashboard_1.0.0
|   artifact.xml
|   appman.json
|   Gadget_1.0.0
|       artifact.xml
|       Test_Gadget
|   Layout_1.0.0
|       artifact.xml
|       Test_Layout

```

1. Create the top level `artifacts.xml` file of the C-App which defines the set of folders included in it as shown below.

```

<artifacts>
    <artifact name="DASTestCApp" version="1.0.0" type="carbon/application">
        <dependency artifact="Eventstore" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
        <dependency artifact="Eventreceiver" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
        <dependency artifact="EventPublisher" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
        <dependency artifact="Eventstream" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
        <dependency artifact="Sparkscripts" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
        <dependency artifact="DataPurging" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
        <dependency artifact="Dashboard" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
        <dependency artifact="Gadget" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
        <dependency artifact="Layout" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
    </artifact>
</artifacts>

```

2. Create separate directories in the top level of the C-App for the above mentioned dependencies (Eventreceiver_1.0.0, EventPublisher_1.0.0 , Eventstore_1.0.0, Eventstream_1.0.0, Sparkscripts_1.0.0 and Data_Purging_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.

3. Create an artifact.xml file inside all dependency directories as follows.

Eventstore_1.0.0 artifact.xml

```
<artifact name="Eventstore" version="1.0.0" type="analytics/eventstore"
serverRole="DataAnalyticsServer">
<file>ORG_WSO2_TEST.xml</file>
</artifact>
```

Eventstream_1.0.0 artifact.xml

```
<artifact name="Eventstream" version="1.0.0" type="event/stream"
serverRole="DataAnalyticsServer">
<file>org.wso2.test_1.0.0.json</file>
</artifact>
```

Eventreceiver_1.0.0 artifact.xml

```
<artifact name="Eventreceiver" version="1.0.0" type="event/receiver"
serverRole="DataAnalyticsServer">
<file>TextWso2EventReceiver.xml</file>
</artifact>
```

EventPublisher_1.0.0 artifact.xml

```
<artifact name="EventPublisher" version="1.0.0" type="event/publisher"
serverRole="DataAnalyticsServer">
<file>TestLoggerPublisher.xml</file>
</artifact>
```

Sparkscripts_1.0.0 artifact.xml

```
<artifact name="Sparkscripts" version="1.0.0" type="analytics/spark"
serverRole="DataAnalyticsServer">
<file>sample_script.xml</file>
</artifact>
```

DataPurging_1.0.0 artifact.xml

```
<artifact name="DataPurging" version="1.0.0" type="analytics/dataPurging"
serverRole="DataAnalyticsServer">
    <file>data_purging.xml</file>
</artifact>
```

Dashboard_1.0.0 artifact.xml

```
<artifact name="Dashboard" version="1.0.0" type="dashboards/dashboard"
serverRole="DataAnalyticsServer">
    <file>appman.json</file>
</artifact>
```

Gadget_1.0.0 artifact.xml

```
<artifact name="Gadget" version="1.0.0" type="dashboards/gadget"
serverRole="DataAnalyticsServer">
    <file>Test_Gadget</file>
</artifact>
```

Layout_1.0.0 artifact.xml

```
<artifact name="Layout" version="1.0.0" type="dashboards/layout"
serverRole="DataAnalyticsServer">
    <file>Test_Layout</file>
</artifact>
```

4. Create the actual artifact which was specified in the `artifact.xml` above. For example, you need to create an event store configuration named `ORG_WSO2_TEST.xml`, an event stream named `org.wso2.t est_1.0.0.json`, an event receiver configuration named `TextWso2EventReceiver.xml`, a Spark script configuration named `sample_script.xml`, a dashboard configuration named `appman.json`, a gadget configuration named `gadget.json`, and a layout configuration named `layout.json`. See below for each of the configuration files.

Stream - org.wso2.test_1.0.0.json

```
{  
    "name": "org.wso2.test",  
    "version": "1.0.0",  
    "nickName": "Test Stream",  
    "description": "A test stream",  
    "metaData": [  
        {  
            "name": "remote_host",  
            "type": "STRING"  
        },  
        {  
            "name": "tenant_id",  
            "type": "INT"  
        },  
        {  
            "name": "activity_id",  
            "type": "STRING"  
        },  
        {  
            "name": "operation_name",  
            "type": "STRING"  
        },  
        {  
            "name": "service_name",  
            "type": "STRING"  
        }  
    ]  
}
```

Event Store - ORG_WSO2_TEST.xml

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<EventStoreConfiguration>
<Source>
    <StreamId>org.wso2.test:1.0.0</StreamId>
</Source>
<TableSchema>
    <ColumnDefinition>
        <Name>meta_remote_host</Name>
        <EnableIndexing>true</EnableIndexing>
        <IsPrimaryKey>false</IsPrimaryKey>
        <EnableScoreParam>false</EnableScoreParam>
        <Type>STRING</Type>
    </ColumnDefinition>
    <ColumnDefinition>
        <Name>meta_tenant_id</Name>
        <EnableIndexing>false</EnableIndexing>
        <IsPrimaryKey>false</IsPrimaryKey>
        <EnableScoreParam>false</EnableScoreParam>
        <Type>INTEGER</Type>
    </ColumnDefinition>
    <ColumnDefinition>
        <Name>correlation_activity_id</Name>
        <EnableIndexing>true</EnableIndexing>
        <IsPrimaryKey>false</IsPrimaryKey>
        <EnableScoreParam>false</EnableScoreParam>
        <Type>FACET</Type>
    </ColumnDefinition>
    <ColumnDefinition>
        <Name>operation_name</Name>
        <EnableIndexing>true</EnableIndexing>
        <IsPrimaryKey>false</IsPrimaryKey>
        <EnableScoreParam>false</EnableScoreParam>
        <Type>STRING</Type>
    </ColumnDefinition>
    <ColumnDefinition>
        <Name>service_name</Name>
        <EnableIndexing>false</EnableIndexing>
        <IsPrimaryKey>false</IsPrimaryKey>
        <EnableScoreParam>false</EnableScoreParam>
        <Type>STRING</Type>
    </ColumnDefinition>
</TableSchema>
</EventStoreConfiguration>

```

Event receiver - TestWso2EventReceiver.xml

```

<eventReceiver name="TestWso2EventReceiver" 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="org.wso2.test" version="1.0.0"/>
</eventReceiver>

```

Event publisher - TestLoggerPublisher.xml

```
<eventPublisher name="TestLoggerPublisher" statistics="disable"
    trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
    <from streamName="org.wso2.test" version="1.0.0"/>
    <mapping customMapping="disable" type="json"/>
    <to eventAdapterType="logger">
        <property name="uniqueId">TestLoggerPublisher</property>
    </to>
</eventPublisher>
```

Spark script - sample_script.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Analytics>
    <Name>AddNewScriptTestWithouTask</Name>
    <Script>define table TEST_CAPP (server_name string, ip STRING, tenant INTEGER,
sequence LONG, summary STRING);SELECT ip FROM TEST_CAPP;SELECT server_name,
count(*) FROM TEST_CAPP GROUP BY server_name;</Script>
    <CronExpression>0 * * * * ?</CronExpression>
</Analytics>
```

Data purging script

```
<analytics-data-purging>
    <cron-expression>0 0 0 * * ?</cron-expression>
    <purge-include-tables>
        <table>org.wso2.test</table>
        <table>TEST_CAPP</table>
    </purge-include-tables>
    <data-retention-days>1</data-retention-days>
</analytics-data-purging>
```

Dashboard - appman.json

```
{
  "id": "appman", "title": "appman", "description": "", "permissions": {"viewers": [], "editors": ["Internal/everyone"]}, "pages": [{"id": "landing", "title": "My Dashboard", "layout": {"id": "layout-3", "title": "Layout 3", "description": "This is a sample grid", "thumbnail": "local://store/layout/layout-3/index.jpg", "url": "local://store/layout/layout-3/index.hbs", "content": "\n      <div class=\"row\">\n        <div id=\"a\" class=\"col-md-4 ues-component-box\"></div>\n        <div id=\"b\" class=\"col-md-4 ues-component-box\"></div>\n        <div id=\"c\" class=\"col-md-4 ues-component-box\"></div>\n      </div>\n      <div id=\"d\" class=\"col-md-10 ues-component-box\"></div>\n      <div id=\"e\" class=\"col-md-2 ues-component-box\"></div>\n      <div class=\"row\">\n        <div id=\"f\" class=\"col-md-4 ues-component-box\"></div>\n        <div id=\"g\" class=\"col-md-4 ues-component-box\"></div>\n      </div>\n    <t><div id=\"h\" class=\"col-md-4 ues-component-box\"></div>\n</t>\n  </div>\n", "content": {"d": [{"id": "24w6ukj9olz69a4i", "content": {"id": "Temperatur e_By_City", "title": "Temperature By City", "type": "gadget", "thumbnail": "local://store/gadget/usa-business-revenue/inde x.png", "data": {"url": "local://store/gadget/Temperature_By_City/index.xml"}, "styles": {"title": "Temperature By City", "borders": true}, "options": {"dataSource": {"type": "STRING", "title": "Data Source", "value": "/portal/gadgets/bar-chart/datasource/dataFile4.jag"}, "options": [], "required": false}, "updateGraph": {"type": "STRING", "title": "Update Interval (s)", "value": "No", "options": [], "required": false}}}], "landing": "landing"
}
```

Gadget - *gadget.json*

```
{
  "id": "Test_Gadget", "title": "Temperature By City", "type": "gadget", "thumbnail": "local://store/gadget/usa-business-revenue/index.png", "data": {"url": "local://store/gadget/Temperature_By_City/index.xml"}}

```

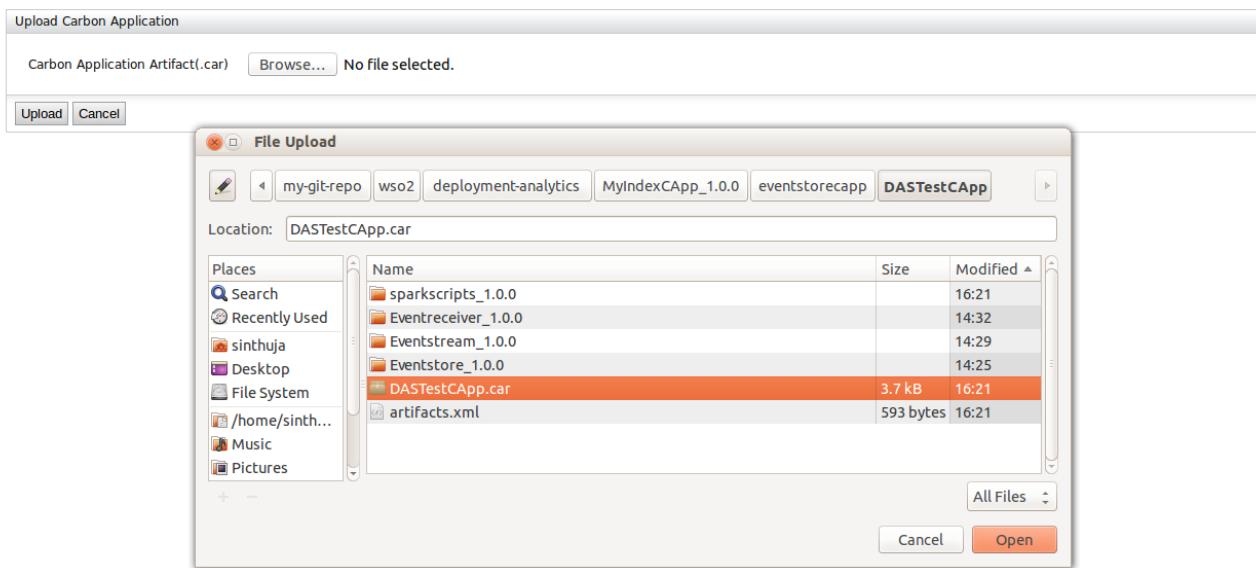
Layout - *layout.json*

```
{
  "id": "Test_Layout",
  "title": "Test_Layout",
  "description": "This is a sample grid",
  "thumbnail": "local://store/layout/layout-3/index.jpg",
  "url": "local://store/layout/layout-3/index.hbs"
}
```

Deploying a C-App

Follow the steps below to deploy a C-App.

1. Log in to WSO2 CEP/DAS 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.



It will take few seconds up to 10 seconds by default to deploy. You view below logs in the CEP/DAS server.

```
[2015-05-26 18:05:44,541] INFO
{org.wso2.carbon.application.deployer.internal.ApplicationManager} - Deploying
Carbon Application : DASTestCApp.car...
[2015-05-26 18:05:45,831] INFO
{org.wso2.carbon.event.stream.core.EventStreamDeployer} - Stream definition is
deployed successfully : org.wso2.test:1.0.0
[2015-05-26 18:05:45,849] INFO
{org.wso2.carbon.event.input.adapter.core.internal.CarbonInputAdapterRuntime} -
Connecting receiver TextWso2EventReceiver
[2015-05-26 18:05:45,852] INFO
{org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to
the junction. Stream:org.wso2.test:1.0.0
[2015-05-26 18:05:45,853] INFO
{org.wso2.carbon.event.receiver.core.EventReceiverDeployer} - Event Receiver
configuration successfully deployed and in active state : TextWso2EventReceiver
[2015-05-26 18:05:45,853] INFO
{org.wso2.carbon.analytics.eventsink.AnalyticsEventStoreDeployer} - Deploying
analytics event store :ORG_WSO2_TEST.xml
[2015-05-26 18:05:45,885] INFO
{org.wso2.carbon.event.stream.core.internal.EventJunction} - WSO2EventConsumer
added to the junction. Stream:org.wso2.test:1.0.0
[2015-05-26 18:05:45,886] INFO
{org.wso2.carbon.analytics.eventsink.AnalyticsEventStoreDeployer} - Deployed
successfully analytics event store :ORG_WSO2_TEST.xml
[2015-05-26 18:05:45,889] INFO
{org.wso2.carbon.analytics.spark.core.SparkScriptCAppDeployer} - Deploying spark
script: sample_script.xml for tenant : -1234
[2015-05-26 18:05:45,932] INFO
{org.wso2.carbon.application.deployer.internal.ApplicationManager} -
Successfully Deployed Carbon Application : DASTestCApp_1.0.0 {super-tenant}
```

4. Click **Main**, then click **Carbon Applications**, and then click **List view**, to see the uploaded Carbon application as shown below.

The screenshot shows the 'Carbon Applications List' page. At the top, there is a breadcrumb navigation: Home > Manage > Carbon Applications > List. On the right side, there is a 'Help' link. Below the header, the text '1 Running Carbon Applications.' is displayed. A table follows, with columns: 'Carbon Applications', 'Version', and 'Actions'. There is one row in the table with the values 'DASTestCApp', '1.0.0', and a set of buttons for 'Delete' and 'Download'. The 'Download' button is highlighted with a red border.

Carbon Applications	Version	Actions
DASTestCApp	1.0.0	Delete Download

5. If you want to download a Carbon application, click **Download** as shown below.

This screenshot is identical to the one above, showing the 'Carbon Applications List' page. It displays one running Carbon application named 'DASTestCApp' with version '1.0.0'. The 'Download' button in the 'Actions' column is highlighted with a red border, indicating it is the target of the user's action.

When SSO is enabled for WSO2 DAS 3.0.1, it is not possible to download Carbon applications by default. If you want to download a Carbon application while SSO is enabled, do the following configurations.

- Comment the following `ServerURL` entry in the `<DAS_HOME>/repository/conf/carbon.xml` file as shown below.

```
<!--ServerURL>local:${carbon.context}/services/</ServerURL-->
```

- Uncomment the following entry in the same file.

```
<ServerURL>https://${carbon.local.ip}:${carbon.management.port}${carbon.context}/services/</ServerURL>
```

6. Click on the **Delete** option to delete the Carbon application as shown below.

Home > Manage > Carbon Applications > List

Carbon Applications List

1 Running Carbon Applications.

Carbon Applications	Version	Actions
DASTestCApp	1.0.0	Delete Download

Also, this removes the Carbon application within 10 seconds, and you view below logs in the server side during the successful undeployment.

```
[2015-05-26 18:06:46,024] INFO
{org.wso2.carbon.application.deployer.internal.ApplicationManager} - Undeploying
Carbon Application : DASTestCApp_1.0.0...
[2015-05-26 18:06:46,030] INFO
{org.wso2.carbon.event.stream.core.EventStreamDeployer} - Stream Definition was
undeployed successfully : org.wso2.test_1.0.0.json
[2015-05-26 18:06:46,033] INFO
{org.wso2.carbon.event.receiver.core.EventReceiverDeployer} - Event Receiver
undeployed successfully : TextWso2EventReceiver.xml
[2015-05-26 18:06:46,034] INFO
{org.wso2.carbon.event.receiver.core.EventReceiverDeployer} - Event receiver
deployment held back and in inactive state :TextWso2EventReceiver.xml, Stream
validation exception :Stream org.wso2.test:1.0.0 does not exist
[2015-05-26 18:06:46,034] INFO
{org.wso2.carbon.event.receiver.core.internal.CarbonEventReceiverService} -
Event receiver : TextWso2EventReceiver in inactive state because event stream
dependency could not be found : org.wso2.test:1.0.0
[2015-05-26 18:06:46,035] INFO
{org.wso2.carbon.event.receiver.core.EventReceiverDeployer} - Event Receiver
undeployed successfully : TextWso2EventReceiver.xml
[2015-05-26 18:06:46,035] INFO
{org.wso2.carbon.analytics.eventsink.AnalyticsEventStoreDeployer} - Undeploying
analytics event store :
/home/sinthuja/projects/my-git-repo/wso2/product-bam/modules/distribution/target/
wso2das-3.0.0-SNAPSHOT/tmp/carbonapps/-1234/1432643744542DASTestCApp.car/Eventsto
re_1.0.0/ORG_WSO2_TEST.xml
[2015-05-26 18:06:46,036] INFO
{org.wso2.carbon.analytics.eventsink.AnalyticsEventStoreDeployer} - Undeployed
successfully analytics event store :
/home/sinthuja/projects/my-git-repo/wso2/product-bam/modules/distribution/target/
wso2das-3.0.0-SNAPSHOT/tmp/carbonapps/-1234/1432643744542DASTestCApp.car/Eventsto
re_1.0.0/ORG_WSO2_TEST.xml
[2015-05-26 18:06:46,036] INFO
{org.wso2.carbon.analytics.spark.core.SparkScriptCAppDeployer} - Undeploying
spark script :
/home/sinthuja/projects/my-git-repo/wso2/product-bam/modules/distribution/target/
wso2das-3.0.0-SNAPSHOT/tmp/carbonapps/-1234/1432643744542DASTestCApp.car/sparkscr
ipts_1.0.0/sample_script.xml for tenant id : -1234
[2015-05-26 18:06:46,042] INFO
{org.wso2.carbon.application.deployer.internal.ApplicationManager} -
Successfully Undeployed Carbon Application : DASTestCApp_1.0.0 {super-tenant}
```

Debugging

This section explains how you can debug WSO2 DAS using the following tools.

- Event Statistics
- Event Tracer
- Siddhi Try It Tool

Event Statistics

Event Statistics is an important feature which helps monitoring purposes of events. This presents realtime requests and responses vs time for all the incoming and outgoing Topics of the CEP/DAS. You can use this visualization to get an idea about system throughput, input frequency and to check whether the inputs are received or outputs are

published. By default, event statistics are disabled in the CEP/DAS to avoid over-head of unnecessary processing of events.

Enabling/Disabling Event Statistics

Even though the Event Statistics feature is enabled in the product, it is not activated by itself. You need to activate event statistics for each and every configuration that you want since monitoring event statistics takes a considerable amount of processing overhead. Change the <StatisticsReporterDisabled> property to false in the <PRODUCT_HOME>/repository/conf/carbon.xml file, to enable event statistics tracing in the product server.

You can enable or disable event statistics for event receivers and event publishers. For an example, follow the steps below to enable event statistics tracing on a event publisher.

Any change in the event statistics status (enable or disable) leads to a redeployment of the necessary configuration.

1. Log in to the product management console using admin/admin credentials.
2. Click **Main**, and then click **Publishers**.
3. Click the **Enable Statistics** option of the corresponding publisher, on which you want to enable tracing as shown below.

The screenshot shows a table with the following data:

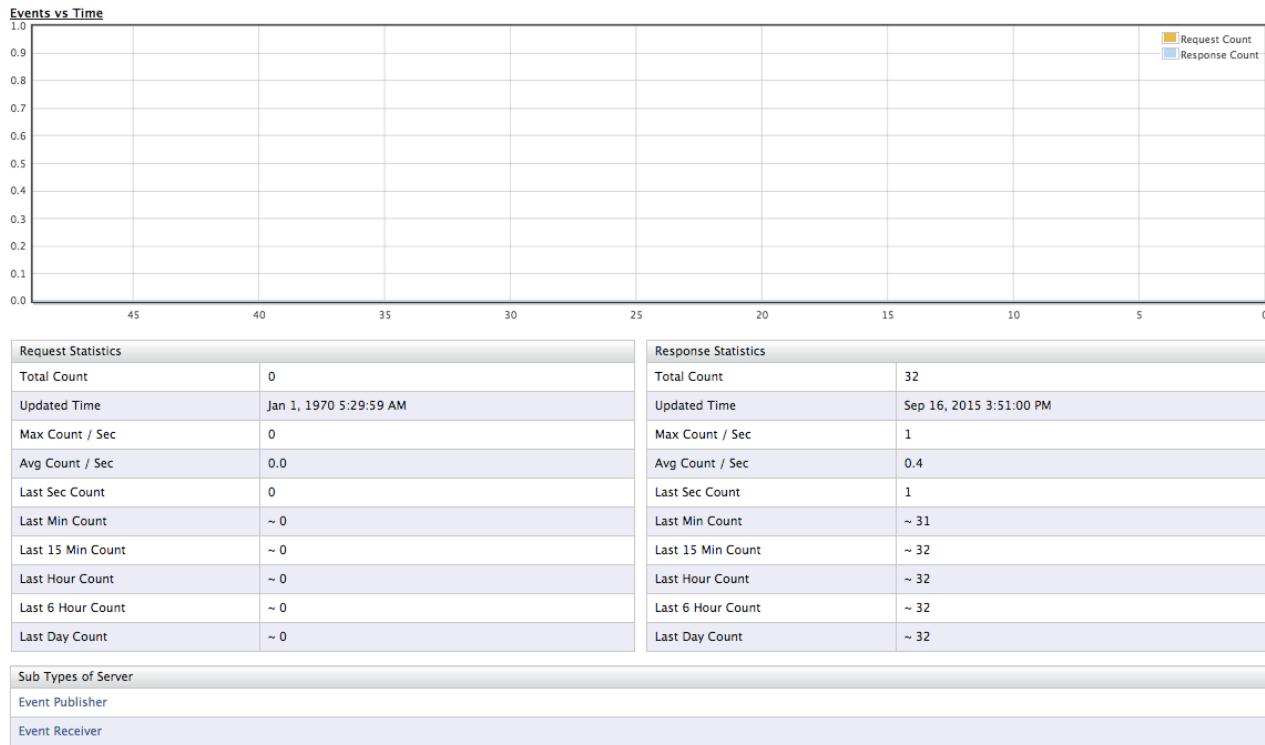
Event Publisher Name	Message Format	Output Event Adapter Type	Input Stream ID	Actions
logger	json	logger	org.wso2.event.sensor.stream:1.0.0	Enable Statistics Enable Tracing Delete Edit

Monitoring Event Statistics

Follow the instructions below to access the Event Tracer.

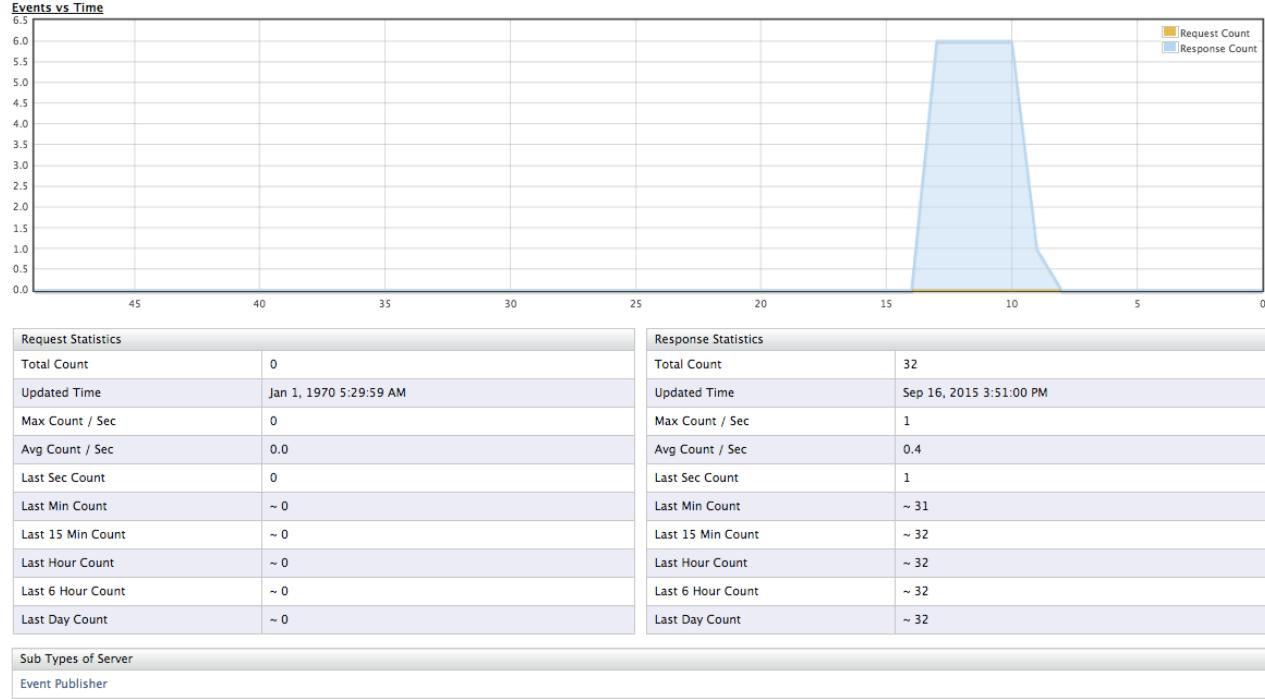
1. Log in to the product management console using admin/admin credentials.
2. Click **Monitor**, and then click **Event Statistics**. If you enabled event statistics for all the configurations (event receivers and event publishers), then you view a window like below.

Event Statistics (All events)



If you send events to the product, then you can get the details of actual statistics details. Then, you view a similar window as shown in the example below.

Event Statistics (All events)



You can further analyse statistics regarding the requests and responses in each configuration in realtime as shown above.

Event Tracer

DAS Event Tracer is an important tool to monitor events. This tool provides huge functionality to trace the event in each and every component. Event tracer will help to check the event when travels along the components. But user needs to enable the tracing in each configuration manually because by default tracing is disabled.

Enabling/Disabling Event Tracer for a Input Event Adapter Configuration

Event Adaptor Name	Event Adaptor Type	Actions
emailAdaptor	email	Enable Statistics Enable Tracing Delete Edit
WSO2EventAdapter	wso2event	Enable Statistics Disable Tracing Delete Edit

As shown above, you can simply enable or disable event tracing for Input Event Adapter configuration. You can follow the sample approach for **Output Event Adapter**, **Event Builder**, **Event Processor** and **Event Formatter** as well, But consider any change in tracing status will lead to redeployment of necessary configuration.

Tracing the events

Follow the instructions below to access the Event Tracer.

1. Click on "Monitor" on the left side to access the "Monitor" menu.
2. In the "Monitor" menu, click on "Event Tracer."
3. The "Event Tracer" page appears.

4. After tracing enabled for necessary configuration, you can see how the incoming events traverse through the components of the DAS and how the events got changed. If you click the **Clear All** button all the event related data will be deleted permanently.

Event Message Tracer

Search Ignore Case

```
<quorepara:CompanyName>Microsoft Corpora</quorepara:CompanyName>
<quodata:QuoteError>false</quodata:QuoteError>
</quodata:StockQuoteEvent>
</quodata:AllStockQuoteStream>
15:02:42,344 [-] [http-nio-9763-exec-1] INFO [Event-Builder] Received event as OMEElement.
<quodata:AllStockQuoteStream xmlns:quodata="http://ws.cdyne.com/">
<quodata:StockQuoteEvent>
<quodata:StockSymbol>MSFT</quodata:StockSymbol>
<quodata>LastTradeAmount>26.36</quodata>LastTradeAmount>
<quodata:StockChange>0.05</quodata:StockChange>
<quodata:OpenAmount>25.05</quodata:OpenAmount>
<quodata:DayHigh>25.46</quodata:DayHigh>
<quodata:DayLow>25.01</quodata:DayLow>
<quodata:StockVolume>20452658</quodata:StockVolume>
<quodata:PrevCls>
<quodata:ChangePercent>0.20</quodata:ChangePercent>
<quodata:FiftyTwoWeekRange>22.73 - 31.58</quodata:FiftyTwoWeekRange>
<quodata:EarnPerShare>2.326</quodata:EarnPerShare>
<quodata:CompanyName>Microsoft Corpora</quodata:CompanyName>
<quodata:QuoteError>false</quodata:QuoteError>
</quodata:StockQuoteEvent>
</quodata:AllStockQuoteStream>
15:02:42,345 [-] [http-nio-9763-exec-1] INFO [Event-Builder] Sending event object array [26.36, MSFT] to all registered basic event listeners
15:02:42,345 [-] [http-nio-9763-exec-1] INFO Events arrived at junction. Event:[26.36, MSFT] Stream:stockQuotes:1.0
15:02:42,345 [-] [http-nio-9763-exec-1] INFO Dispatching events to the siddhi engine. Events: [26.36, MSFT]
15:02:42,346 [-] [http-nio-9763-exec-1] INFO Events arrived at junction. Event:[Event {streamId='newStockQuoteStream', timeStamp=1375349562345, data=[26.36, MSFT], type=new}] Stream:stockStream:1.0.0
15:02:42,346 [-] [http-nio-9763-exec-1] INFO Dispatching events to the event formatter. Events: [Event {streamId='newStockQuoteStream', timeStamp=1375349562345, data=[26.36, MSFT], type=new}]
```

[Clear All](#)

5. Here you can use the search option to refine the data in the UI as shown below.

Event Message Tracer

Search Ignore Case

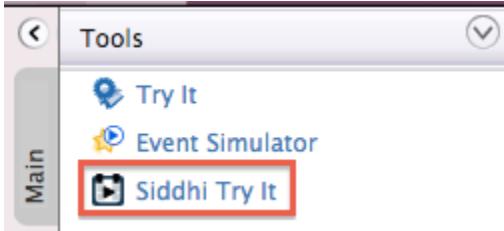
```
15:02:42,346 [-] [http-nio-9763-exec-1] INFO Events arrived at junction. Event:[Event {streamId='newStockQuoteStream', timeStamp=1375349562345, data=[26.36, MSFT], type=new}] Stream:stockStream:1.0.0
15:02:42,346 [-] [http-nio-9763-exec-1] INFO Dispatching events to the event formatter. Events: [Event {streamId='newStockQuoteStream', timeStamp=1375349562345, data=[26.36, MSFT], type=new}]
15:02:42,370 [-] [http-nio-9763-exec-1] INFO Events arrived at junction. Event:[Event {streamId='newStockQuoteStream', timeStamp=1375349562370, data=[36.0, MSFT], type=new}] Stream:stockStream:1.0.0
15:02:42,370 [-] [http-nio-9763-exec-3] INFO Dispatching events to the event formatter. Events: [Event {streamId='newStockQuoteStream', timeStamp=1375349562370, data=[36.0, MSFT], type=new}]
```

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 DAS management console, click **Tools**, and then click **Siddhi Try It** as shown below.



2. Enter the **Execution Plan** as shown in the below example.

Siddhi Try It

Execution Plan

```

1 @Plan:name('TestExecutionPlan')
2
3 define stream sensorStream (sensorId string, temperature float);
4
5 @info(name = 'query1')
6 from sensorStream[temperature>98.6]
7 select sensorId
8 insert into outputStream;

```

The main elements of an execution plan are described below.

For more information on execution plans, see [Working with Execution Plans](#).

Element	Description	Example
Execution plan name	The name of the execution plan. It can contain only alphanumeric characters and '_' character.	TestExecutionPlan
Input stream	The mappings between the available event stream and the input stream of the Siddhi runtime, which is defined inside the query expressions.	sensorStream
Query expressions	<p>The event processing logic written in Siddhi QL.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> When defining more than one query, end each query with a semicolon. Defining a query name (e.g. query1) is optional. </div>	<pre> @info(name = 'query1') from sensorStream[temperature>98.6] select sensorId insert into outputStream; </pre>
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.	outputStream

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.

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](#) .

Event Stream

Begin Time

```
sensorStream=[tempID1,99.8]
delay(100)
sensorStream=[tempID2,80.6]
```

- 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.

Result

```
[ {
  "timestamp": 1432039938174,
  "data": [
    "tempID1"
  ],
  "isExpired": false
}]
```

Managing DAS Artifacts via 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 tool 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 DAS 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.

Stream	Purpose
org.wso2.event.temperature.stream	To calculate sum, average, maximum and minimum values for temperature over a period of time and display the results on a dashboard.
org.wso2.event.pressure.stream	To calculate sum, average, maximum and minimum values for pressure over a period of time and display the results on a dashboard.
org.wso2.event.humidity.stream	To calculate sum, average, maximum and minimum values for humidity over a period of time and display the results on a dashboard.

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 DAS artifacts.

Currently, this feature supports following WSO2 DAS artifact types

- Realtime Analytics - Execution Plans
- Batch Analytics - Spark Scripts
- Event streams
- Event sinks
- Gadgets
- 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 DAS 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 event sinks](#)
- [Step 4: Configure real time analytics](#)
- [Step 5: Configure batch analytics](#)
- [Step 6: Configure an event publisher](#)
- [Step 7: Configure a gadget](#)
- [Step 8: Configure a dashboard](#)
- [Step 9: Configure stream mapping](#)
- [Step 10: Configure parameters](#)
- [Step 11: Configure common artifacts](#)
- [Step 12: 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 `<DAS_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-->
        <script src="/" />
    </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	Purpose	Sub Element
Domain	This is a logical collection of Scenarios. One domain can be used to group related scenarios together for better organization.	<domain name="SensorDataAnalysis">

Scenario	A description of the scenario to be analyzed. e.g., Analyzing average and maximum temperature.	<scenario type="SensorAnalytics">
Template	A templated artifact.	<template type="type">
Event Stream	This defines the event stream definitions associated with the scenario.	<template type="eventstream">
Execution Plan	This defines the Siddhi execution plan based on which the events are processed.	< template type = "realtime" >
Gadget	This defines the gadgets required to view the information processed by the event flows created via the template in a specific format.	< template type = "gadget" >
Dashboard	This defines the dashboards to display the gadgets with the information processed by the event flow.	< template type = "dashboard" >
Stream mapping	This maps an existing input stream in your WSO2 DAS installation with the event stream defined in the template manager.	< streamMappings >
Parameters	This defines configurable variables. The \$ sign is used to indicate configurable fields.	< parameters >
Common artifacts	This allows you to include artifacts that are common to all the scenarios using the templates in the template manager.	< commonArtifacts >
Scripts	To include java script functions that can be used as parameters.	<scripts>

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 and Event Tables](#).

To add required event streams for this use case, add the following configuration under the `<template type="event_stream">` element in the `<DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file you created, and save.

Event Stream Template[Expand source](#)

```
<template type="eventstream">
{
    "name": "org.wso2.event.$sensorType.stream",
    "version": "1.0.0",
    "nickName": "",
    "description": "",
    "payloadData": [
        {
            "name": "sensor_id",
            "type": "STRING"
        },
        {
            "name": "sensor_value",
            "type": "DOUBLE"
        }
    ]
}
</template>
```

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 DAS 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 DAS Management Console and define the required event stream. For detailed instructions, see [Understanding Event Streams and Event Tables](#).
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 event sinks

This step involves adding event sinks to be included in the template. An event sink configuration contains information relating to what attributes need to be persisted in a WSO2 DAS event stream. An event sink configuration is created in the `<DAS_HOME>/repository/deployment/server/eventsink` directory when you persist an event stream. For more information about persisting dreams, see [Persisting Data for Batch Analytics](#).

To add the event sink required for this use case, add the following configuration under the `<template type="event sink">` element in the `<DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file you created, and save.

Event Sink Element[Expand source](#)

```

<template type="eventsink">
<! [CDATA[<EventStoreConfiguration>
    <TableSchema>
        <ColumnDefinition>
            <Name>sensor_type</Name>
            <IsFacet>false</IsFacet>
            <EnableIndexing>false</EnableIndexing>
            <IsPrimaryKey>true</IsPrimaryKey>
            <EnableScoreParam>false</EnableScoreParam>
            <Type>STRING</Type>
        </ColumnDefinition>
        <ColumnDefinition>
            <Name>sensor_id_distinct_count</Name>
            <IsFacet>false</IsFacet>
            <EnableIndexing>false</EnableIndexing>
            <IsPrimaryKey>false</IsPrimaryKey>
            <EnableScoreParam>false</EnableScoreParam>
            <Type>LONG</Type>
        </ColumnDefinition>
        <ColumnDefinition>
            <Name>count</Name>
            <IsFacet>false</IsFacet>
            <EnableIndexing>false</EnableIndexing>
            <IsPrimaryKey>false</IsPrimaryKey>
            <EnableScoreParam>false</EnableScoreParam>
            <Type>LONG</Type>
        </ColumnDefinition>
        <ColumnDefinition>
            <Name>sum</Name>
            <IsFacet>false</IsFacet>
            <EnableIndexing>false</EnableIndexing>
            <IsPrimaryKey>false</IsPrimaryKey>
            <EnableScoreParam>false</EnableScoreParam>
            <Type>DOUBLE</Type>
        </ColumnDefinition>
        <ColumnDefinition>
            <Name>average</Name>
            <IsFacet>false</IsFacet>
            <EnableIndexing>false</EnableIndexing>
            <IsPrimaryKey>false</IsPrimaryKey>
            <EnableScoreParam>false</EnableScoreParam>
            <Type>DOUBLE</Type>
        </ColumnDefinition>
        <ColumnDefinition>
            <Name>max</Name>
            <IsFacet>false</IsFacet>
            <EnableIndexing>false</EnableIndexing>
            <IsPrimaryKey>false</IsPrimaryKey>
            <EnableScoreParam>false</EnableScoreParam>
            <Type>DOUBLE</Type>
        </ColumnDefinition>
        <ColumnDefinition>
            <Name>min</Name>
            <IsFacet>false</IsFacet>
            <EnableIndexing>false</EnableIndexing>
            <IsPrimaryKey>false</IsPrimaryKey>
            <EnableScoreParam>false</EnableScoreParam>
        </ColumnDefinition>
    </TableSchema>
</EventStoreConfiguration>]]>

```

```
<Type>DOUBLE</Type>
</ColumnDefinition>
</TableSchema>
<Source>
<StreamId>org.wso2.event.$sensorType.statistics.stream:1.0.0</StreamId>
</Source>
<MergeSchema>false</MergeSchema>
```

```

<RecordStoreName>EVENT_STORE</RecordStoreName>
</EventStoreConfiguration>]]>
</template>

```

The following procedure is an easy approach to create the configuration for an event sink to be added to the Template Manager.

1. Log into the WSO2 DAS Management Console and define the required event stream. For detailed instructions, see [Understanding Event Streams and Event Tables](#).
2. Persist the event stream you created. For detailed instructions, see [Persisting Data for Batch Analytics](#).
3. Open the <DAS_HOME>/repository/deployment/server/eventsink/<Event_Stream_Name>.xml file. Copy the configuration in this file and add it to your template under the <template type="eventsink"> element.
4. Add the required configurable parameters and save.

Step 4: 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 <DAS_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

[Expand source](#)

```

<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 \$timeInMin s 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 DAS 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 DAS 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 <DAS_HOME>/repository/conf/template-manager/domain-template directory.
3. Add the required configurable parameters and save.

Step 5: Configure batch analytics

This step involves adding the Spark scripts required for this use case to the template. The Batch Analytics functionality of DAS is powered by Apache Spark. Therefore, Spark scripts are run in order to perform the required queries on persisted data. For more information about Spark scripts, see [Scheduling Batch Analytics Scripts](#).

To add the batch analytics configuration required for this use case, add the following configuration under the <template type="batch" > element in the <DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml file.

An Batch Analytics configuration should always be added inside a CDATA element.

Batch Analytics Element[Expand source](#)

```

<template type="batch">
    <executionParameters>
        <cron>0 0 12 1/7 * ?*</cron>

        <sparkScript>
            <![CDATA[
                create temporary table accessTokenRefreshTime using CarbonAnalytics
                options (tableName "ORG_WSO2_ANALYTICS_APIM_ACESSTOKENREFRESHTIMEDIFFERENCE", schema
                "userId STRING, clientId STRING, scopes STRING, timeDifference LONG, timestamp LONG");

                create temporary table accessTokenRefreshAvrgTime using CarbonAnalytics options
                (tableName "ORG_WSO2_ANALYTICS_APIM_ACESSTOKENREFRESHSUMMARYTABLE", schema "userId
                STRING -i, clientId STRING -i, scopes STRING -i, minTimeDifference DOUBLE -i,
                maxTimeDifference DOUBLE -i", primaryKeys "userId, clientId, scopes");

                INSERT INTO TABLE accessTokenRefreshAvrgTime
                    SELECT temp.userId, temp.clientId, temp.scopes,
                    getpercentileValue(AVG(timeDifference), SQRT(AVG(timeDifference*timeDifference) -
                    AVG(timeDifference)*AVG(timeDifference))), $lowerPercentile) as minTimeDifference,
                    getpercentileValue(AVG(timeDifference), SQRT(AVG(timeDifference*timeDifference) -
                    AVG(timeDifference)*AVG(timeDifference))), $upperPercentile) as maxTimeDifference
                    FROM
                    (SELECT userId, clientId,scopes, timeDifference
                    FROM accessTokenRefreshTime
                    WHERE timestamp >= offsetInDays(-7)) temp
                    GROUP BY userId, clientId, scopes;

            ]]>
            </sparkScript>

        </executionParameters>
    </template>

```

Step 6: Configure an event publisher

This step involves adding an event publisher to the template. Event publishers publish events from WSO2 DAS to external consumers. For more information about event publishers, see [Creating Alerts - Creating Event Publishers](#).

To add the event publisher configuration required for this use case, add the following configuration under the `<template type="eventPublisher">` element in the `<DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

An event receiver configuration should always be added inside a CDATA element.

Event Publisher Element[Expand source](#)

```
<template type="event publisher">
<![CDATA[<eventPublisher name="$sensorType.statistics.stream.publisher"
    statistics="disable" trace="disable"
    xmlns="http://wso2.org/carbon/eventpublisher">
    <from streamName="org.wso2.event.$sensorType.statistics.stream" version="1.0.0"/>
    <mapping customMapping="disable" type="wso2event"/>
    <to eventAdapterType="ui"/>
</eventPublisher>]]>
</template>
```

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 DAS Management Console and create the required event publisher. For detailed instructions to create an event publisher, see [Creating 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 `<template type="eventPublisher">` element of your template file in the `<DAS_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 7: Configure a gadget

This step involves adding a gadget configuration to the template. For more information about gadgets, see [Adding Gadgets to a Dashboard](#).

To add the gadget configuration required for this use case, add the following configuration under the `< template type = "gadget" >` element in the `<DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

Gadget Element[Expand source](#)

```

<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[var getGadgetLocation =
function(){
    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 `<DAS_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 generate this content.

1. Create a new gadget as required for the scenario. For detailed instructions, see [Adding Gadgets to a Dashboard](#).
2. The artifacts mentioned above are located in the `<DAS_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 `<DAS_HOME>/repository/conf/template-manager/domain-template` directory.

3. Add the required configurable parameters and save.

Step 8: 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](#).

To add the dashboard configuration required for this use case, add the following configuration under the `<template type = "dashboard" >` element in the `<DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

Dashboard element

[Expand source](#)

```
<template type="dashboard">
<config>
    <properties>
        <property name="dashboardId">analytics-$sensorType-dashboard</property>
    </properties>
    <content><![CDATA[ {
        "accessTokenUrl": "",
        "apiKey": "",
        "apiSecret": "",
        "banner": {
            "customBannerExists": false,
            "globalBannerExists": false
        },
        "defaultPriority": "5",
        "description": "",
        "hideAllMenuItems": false,
        "id": "analytics-$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": ""
            },
            "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": {
    "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"
```

```

        }]></content>
    </config>
</template>

```

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 [Adding a Dashboard](#).

The dashboard configuration is saved in the `/_system/config/ues/dashboards/<Dashboard_Name>` registry path. To access the registry, log into the WSO2 DAS 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 `<DAS_HOME>/repository/conf/template-manager/domain-template` directory.

Step 9: 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 DAS 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` 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 `<DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

```

<streamMappings>
    <streamMapping to="org.wso2.event.$sensorType.stream:1.0.0" />
</streamMappings>

```

Step 10: 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 `<DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

Parameters[Expand source](#)

```
<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 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 `<commonArtifacts>` element to the `<DAS_HOME>/repository/conf/template-manager/domain-template/Sensor-Analytics.xml` file.

Common Artifacts[Expand source](#)

```
<commonArtifacts>
    <artifact type="eventstream">
        <!--This Stream can be used across the whole Domain.-->
        {
            "name": "commonStream",
            "version": "1.0.0",
            "nickName": "",
            "description": "",
            "payloadData": [
                {
                    "name": "timestamp",
                    "type": "LONG"
                },
                {
                    "name": "value",
                    "type": "DOUBLE"
                }
            ]
        }
    </artifact>
</commonArtifacts>
```

Step 12: Configure Scripts

A script can be included either using an external file that is stored in the `<DAS_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.

Scripts

› [Expand source](#)

```
<scripts>
<!--This script points to the
<DAS_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.

Event Publisher Using Scripts

› [Expand source](#)

```
<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 `<DAS_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](#).

This tutorial uses the `SensorAnalytics` template that is shipped by default in the `<DAS_HOME>/repository/conf/template-manager/domain-template` directory.

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://<DAS_HOST>:<DAS_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.

The screenshot shows the WSO2 Template Manager dashboard. At the top, there is a header bar with the WSO2 logo and the text "Template Manager". On the right side of the header, it says "admin@carbon.super" with a user icon. Below the header, the word "Domains" is centered. A message "Select a Domain to proceed" is displayed. There are two green rectangular cards representing domains. The first card is for "SensorAnomalyDetection" with the subtext "Detecting anomalies in sensor reading". The second card is for "SensorDataAnalysis" with the subtext "Domain for sensor data analysis".

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.

The screenshot shows the "Deployed Scenarios" page for the "SensorDataAnalysis" domain. At the top, there is a header bar with the WSO2 logo and the text "Template Manager / SensorDataAnalysis". On the right side of the header, it says "admin@carbon.super" with a user icon. Below the header, the title "Deployed Scenarios" is centered. Underneath the title, there is a breadcrumb navigation "Template Manager / SensorDataAnalysis". A prominent green button at the top of the main content area is labeled "Create New Scenario". Below this button, the text "No Scenarios to be listed" is displayed.

- Enter values in the **Edit Scenario** page as follows.

Field	Value
Scenario Type	<p>SensorAnalytics</p> <p>This is the template created by following the instructions in Configuring a Template for Template Manager.</p>
Scenario Name	SensorAnalytics
Description	Calculate maximum and average temperature values.

Enter values for the configurable parameters as shown below.

Configurable Parameters	Value
Time(Mins)	1
Sensor Type Name	temperature

- 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.

The screenshot shows the WSO2 Template Manager interface. At the top, there is a header bar with the WSO2 logo, the text "Template Manager", and a user profile icon. Below the header, the title "Deployed Scenarios" is displayed. Underneath the title, there is a breadcrumb navigation bar with the text "Template Manager / SensorDataAnalysis". A green button labeled "Create New Scenario" is visible. The main content area contains a table with one row, showing the details of the "SensorAnalytics" scenario. The table columns are "Scenario Name", "Description", "Scenario Type", and "Actions". The "Actions" column includes links for "Delete" and "Edit".

Deployed Scenarios

Template Manager / SensorDataAnalysis

Create New Scenario

Scenario Name	Description	Scenario Type	Actions	
SensorAnalytics	Calculate maximum and average temperature values.	SensorAnalytics	Delete	Edit

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**.

STREAM MAPPING

Mapped From: Stream Name

Select an input stream to map

- ✓ AlertsNotifications:1.0.0
- fusedSpatialEvent:1.0.0
- org.wso2.event.temperature.statistics.stream:1.0.0
- org.wso2.event.temperature.stream:1.0.0
- processedSpatialEvents:1.0.0
- rawInputStream:1.0.0
- standardSpatialEvents:1.0.0

Save Mapping

Once this configuration is successfully completed, the artifacts included in the template are deployed in WSO2 DAS. You can view the event flow for the template artifacts by logging into the WSO2 DAS Management Console, and clicking **Main => Flow** where it is displayed as follows.



6. In the WSO2 DAS 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.

DASHBOARDS

analytics-temperature-dashboard URL: analytics-temperature-dashboard	Geo Dashboard URL: geo-dashboard
View Design Settings	View Design Settings

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 CSV File](#).
 - Download the `events.csv` file from [here](#) and save it in a preferred location
 - In the WSO2 DAS Management Console, click **Tools** and then click **Event Simulator** to open the **Event Stream Simulator** page.

- c. Click **Choose File**, and browse for the events.csv file you downloaded in sub step a. Then click **Upload** to upload the file. Click **OK** in the message that appears to confirm that the file is uploaded, and refresh the page.
- d. Click **Configure** for the events.csv file to open the **Event Mapping Configuration** dialog box. Enter the following values in this dialog box.

Parameter	Value
Select the target event stream	org.wso2.event.temperature.stream:1.0.0
Field delimiter	,

Click **Configure** and then click **OK** to close the message that appears to confirm that the configuration is successful.

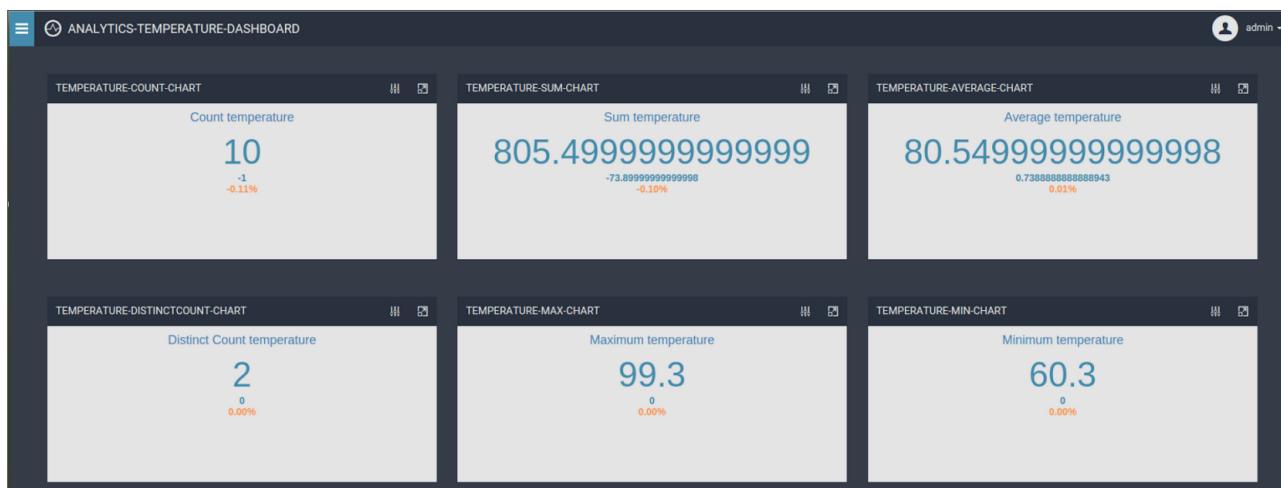
- e. Click **Play** to simulate the event flow.

The screenshot shows the WSO2 Data Analytics Server's Event Stream Simulator interface. At the top, there are navigation links: Home > Tools > Event Simulator and a Help link. Below the navigation is a title bar: Event Stream Simulator. The interface is divided into two main sections:

- Send single event:** This section contains fields for "Event Stream Name" (with a dropdown menu "select event stream") and "Stream Attributes". It includes "Send" and "Clear" buttons.
- Send multiple events:** This section has a table titled "Input Data by File" with a "switch to configure database for simulation" link. The table columns are: File, Stream Configuration, Delay between events(ms), and Action. A row is shown for "events.csv" with "click the configure button" in all three columns. The "Action" column contains "Configure" and "Delete" buttons, with "Configure" being highlighted. Below the table are "Choose File" and "upload" buttons.

A status message "Reserved." is displayed at the bottom of the page.

- 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 information useful for system administrators.

- [Spark Configurations](#)
- [Enabling/Disabling Selected DAS Components](#)
- [Purging Data](#)
- [Analytics Migration Tool](#)
- [Analytics Data Backup / Restore Tool](#)
- [Performance Tuning](#)
- [Configuration Guide](#)
- [Connecting a DAS Instance to an Existing External Apache Spark Cluster](#)
- [Storing Index Data](#)
- [Configuring Single Sign-On for WSO2 DAS](#)
- [Deployment and Clustering](#)
- [Working with Product Specific Analytics Profiles](#)
- [Supporting Different Transports](#)
- [Installing WSO2 GPL Features](#)
- [Changing the Host Name](#)
- [Product Administration](#)

Spark Configurations

This section covers the configurations required to use Apache Spark with WSO2 DAS.

- [Adding jar files to the Spark classpath](#)
- [Carbon related configurations](#)
- [Default Spark related configurations](#)
- [Optional Spark related configurations](#)

Adding jar files to the Spark classpath

When starting the Spark Driver Application in the DAS server, Spark executors are created within the same node. The following jars are included in the classpath for these executors by default.

- apache-zookeeper
- axiom
- axis2
- axis2-json
- cassandra-thrift
- chill
- com.datastax.driver.core
- com.fasterxml.jackson.core.jackson-annotations
- com.fasterxml.jackson.core.jackson-core
- com.fasterxml.jackson.core.jackson-databind
- com.fasterxml.jackson.module.jackson.module.scala
- com.google.gson
- com.google.guava
- com.google.protobuf
- com.jayway.jsonpath.json-path
- com.ning.compress-lzf
- com.sun.jersey.jersey-core
- com.sun.jersey.jersey-server
- commons-codec
- commons-collections
- commons-configuration
- commons-httpclient
- commons-io

- commons-lang
- config
- h2-database-engine
- hadoop-client
- hazelcast
- hbase-client
- hector-core
- htrace-core
- htrace-core-apache
- httpclient
- httpcore
- io.dropwizard.metrics.core
- io.dropwizard.metrics.graphite
- io.dropwizard.metrics.json
- io.dropwizard.metrics.jvm
- javax.cache.wso2
- javax.servlet.jsp-api
- jaxb
- jdbc-pool
- jdom
- jettison
- json
- json-simple
- json4s-jackson
- kryo
- libthrift
- lucene
- mesos
- minlog
- net.minidev.json-smart
- netty-all
- objenesis
- org.apache.commons.lang3
- org.apache.commons.math3
- org.jboss.netty
- org.roaringbitmap.RoaringBitmap
- org.scala-lang.scala-library
- org.scala-lang.scala-reflect
- org.spark-project.protobuf.java
- org.spark.project.akka.actor
- org.spark.project.akka.remote
- org.spark.project.akka.slf4j
- org.wso2.carbon.analytics.api
- org.wso2.carbon.analytics.dataservice.commons
- org.wso2.carbon.analytics.dataservice.core
- org.wso2.carbon.analytics.datasource.cassandra
- org.wso2.carbon.analytics.datasource.commons
- org.wso2.carbon.analytics.datasource.core
- org.wso2.carbon.analytics.datasource.hbase
- org.wso2.carbon.analytics.datasource.rdbms
- org.wso2.carbon.analytics.eventsink
- org.wso2.carbon.analytics.eventtable
- org.wso2.carbon.analytics.io.commons
- org.wso2.carbon.analytics.spark.core
- org.wso2.carbon.analytics.stream.persistence
- org.wso2.carbon.base
- org.wso2.carbon.cluster.mgt.core

- org.wso2.carbon.core
- org.wso2.carbon.core.common
- org.wso2.carbon.core.services
- org.wso2.carbon.databridge.agent
- org.wso2.carbon.databridge.agent
- org.wso2.carbon.databridge.commons

In addition any jars available in the <DAS_HOME>/repository/conf/lib directory are also appended to the class path.

If you want to add additional jars, you can add them to the SPARK_CLASSPATH in the <DAS_HOME>/bin/external-spark-classpath.conf file in a UNIX environment.

Each path should have a separate line.

When WSO2 DAS connects with an external Spark cluster, the distribution of DAS is copied to each node in the Spark cluster. This allows each Spark node to access the jars it needs to work with WSO2 DAS.

Carbon related configurations

Following are the Carbon related configurations that are used for Apache Spark. These configurations are shipped with the product by default in the <DAS_home>/repository/conf/analytics/spark/spark-defaults.conf file.

Property	Default Value	Description
carbon.spark.master	local	<p>The Spark master has three possible states as follows:</p> <ul style="list-style-type: none"> • local: This starts Spark in the local mode. e.g, carbon.spark.master local or carbon.spark.master local[2] • client: This mode results in the DAS acting as a client for an external Spark cluster. e.g., carbon.spark.master spark://<host name>:<port>. For more details on setting up WSO2 DAS and Apache Spark in this mode, see Connecting a DAS Instance to an Existing External Apache Spark Cluster • cluster: This mode results in the DAS creating its own Spark cluster using Carbon Clustering. When Spark runs in this mode, it is required to specify a value for the carbon.spark.master.count property. e.g., carbon.spark.master local AND carbon.spark.master.count <number of redundant masters>
carbon.spark.master.count	1	<p>The maximum number of masters allowed at a given time when DAS creates its own Spark cluster.</p> <p>This property is applicable only when the Spark master runs in the cluster mode.</p>

carbon.das.symbolic.link	This links to your DAS home by default.	The symbolic link for the jar files in the Spark class path. In a clustered DAS deployment, the directory path for the Spark Class path is different for each node depending on the location of the <DAS_HOME>. The symbolic link redirects the Spark Driver Application to the relevant directory for each node when it creates the Spark class path. The symbolic link should be located in the same path for each <DAS_HOME>. The symbolic link is not specified by default. When it is not specified, the jar files are added in the DAS home.
--------------------------	-----------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Default Spark related configurations

Following are the Apache Spark related configurations that are used in WSO2 DAS. These configurations are shipped with the product by default in the <DAS_home>/repository/conf/analytics/spark/spark-defaults.conf file.

For more information on the below Spark configuration properties, go to [Apache Spark Documentation](#).

Application configurations

Property	Default Value
spark.app.name	CarbonAnalytics
spark.driver.cores	1
spark.driver.memory	512m
spark.executor.memory	512m

Spark UI configurations

Property	Default Value
spark.ui.port	CarbonAnalytics
spark.history.ui.port	18080

Compression and serialization configurations

Property	Default Value
spark.serializer	org.apache.spark.serializer.KryoSerializer
spark.kryoserializer.buffer	256k
spark.kryoserializer.buffer.max	256m

Networking configurations

Property	Default Value
----------	---------------

spark.blockManager.port	12000
spark.broadcast.port	12500
spark.driver.port	13000
spark.executor.port	13500
spark.filesServer.port	14000
spark.replClassServer.port	14500

Scheduling configurations

Property	Default Value
spark.scheduler.mode	FAIR

In addition to having FAIR as the value for the spark.scheduler.mode property in the spark-defaults.conf file, it is required to have a pool configuration with the schedulingMode parameter set to FAIR in the <DAS_HOME>/repository/conf/analytics/spark/fairscheduler.xml file as shown in the configuration below. This is because Apache Spark by default uses a scheduler pool which runs scheduler tasks in a First-In-First-Out (FIFO) order.

```
<?xml version="1.0"?>

<allocations>
    <pool name="carbon-pool">
        <schedulingMode>FAIR</schedulingMode>
        <weight>1000</weight>
        <minShare>1</minShare>
    </pool>
    <pool name="test">
        <schedulingMode>FIFO</schedulingMode>
        <weight>1</weight>
        <minShare>0</minShare>
    </pool>
</allocations>
```

Standalone cluster configurations

Property	Default Value
spark.deploy.recoveryMode	CUSTOM
spark.deploy.recoveryMode.factory	org.wso2.carbon.analytics.spark.core.deploy.AnalyticsRecoveryMod

Master configurations

Property	Default Value

spark.master.port	7077
spark.master.rest.port	6066
spark.master.webui.port	8081

Worker configurations

Property	Default Value
spark.worker.cores	1
spark.worker.memory	1g
spark.worker.dir	work
spark.worker.port	11000
spark.worker.webui.port	11500

Optional Spark related configurations

The following configurations can be added to the <DAS_home>/repository/conf/analytics/spark/spark-defaults.conf file if you want to limit the space allocated for log files generated and saved in the <DAS_HOME>/work directory.

```
spark.executor.logs.rolling.strategy size
spark.executor.logs.rolling.maxSize 10000000
spark.executor.logs.rolling.maxRetainedFiles 10
```

Property	Description
spark.executor.logs.rolling.strategy	This indicates the strategy used to control the amount of logs saved in the <DAS_HOME>/work directory. In the above configuration, property value size indicates that the amount of logs that are allowed to be kept is restricted based on the size.
spark.executor.logs.rolling.maxSize	The maximum size (in bytes) allowed for logs saved in the <DAS_HOME>/work directory at any given time. Older log files are deleted when new logs are generated so that the specified maximum size is not exceeded.

spark.executor.logs.rolling.maxRetainedFiles	The maximum number of log files allowed to be kept in the <DAS_HOME>/work directory at any given time. Older log files are deleted when new logs are generated so that the specified maximum number of files is not exceeded. In the above configuration, this property is overruled by the spark.executor.logs.rolling.maxSize property because the value specified for the spark.executor.logs.rolling.strategy is size. If the maximum size specified for logs is reached, older logs are deleted even if the maximum number of files specified is not yet reached.
----------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Enabling/Disabling Selected DAS Components

You can enable/disable WSO2 DAS components depending on how you deploy the DAS servers. You can disable some of the components/features that do not need to function in a specific node. This not only enables the DAS nodes to use the resources effectively for the intended operation of the node, but also provides high availability for the selected operations of DAS.

This is done by setting a system property with server startup command as explained below. Therefore, you can simply use that same DAS distribution for all the nodes. Alternatively, you can enable/disable selected components/features by starting up the server using one or more corresponding system properties.

Command (on Linux)	Function	Usage
sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableAnalyticsEngine=true	If this system property is set, then the Spark server will not startup in this node.	You can use this property when you want to have a node only as a receiver node, indexing node, publisher node or real time analytics node.
sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableAnalyticsExecution=true	If this system property is set, then the node does not join the task execution of the Spark scripts that you have scheduled in the cluster, nor you can't execute any Spark scripts from the node.	You can use this property when you want to have a node only as a receiver node, indexing node, publisher node, realtime analytics node, or Spark analyzer node which accepts jobs from a remote server.
sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableIndexing=true	If this system property is set, then the node will not participate in the indexing task.	You can use this property when you want to have a node only as a receiver node, publisher node, realtime analytics node, or Spark analyzer node which accepts jobs from a remote server.

<pre>sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableEventSink=true</pre>	<p>If this system property is set, the events received for the streams will not participate in persisting the events even the stream has been configured to be persisted.</p>	<p>You can use this property when you want to have a node only as a real time analytics node.</p>
<pre>sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableDataPurging=true</pre>	<p>If this system property is set, then that particular node will not join the purging operation. This property is applicable for both global task operations and tasks scheduled through the Data Explorer.</p>	<p>This is only useful in a clustered environment. When you schedule a purging operation, that particular task can be scheduled in any DAS node. But if you want to prevent those purging tasks being scheduled in a particular node such as a dashboard node, then you have to use this property.</p>
<pre>sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableAnalyticsSparkCtx=true</pre>	<p>If this system property is set, then that particular node will not instantiate a Spark context. This means that there will not be a Spark app created in the server startup.</p>	<p>This property allows you to use a DAS cluster as a Spark cluster, and submit Spark apps to it. For an example, WSO2 Machine Learner (ML) can submit its Spark Apps to DAS. For more information on connecting WSO2 ML to an external Spark cluster, see the With external Spark cluster section in the Deployment Patterns page of WSO2 ML documentation.</p>
<pre>sh <PRODUCT_HOME>/bin/wso2server.sh -DenableAnalyticsStats=true</pre>	<p>If this system property is set, then the Spark query execution statistics are printed in the Carbon Console.</p>	<p>This property can be used when you need to view the Analytics statistics in the Carbon Console.</p>
<pre>s h <PRODUCT_HOME>/bin/wso2server.sh -DenableIndexingStats=true</pre>	<p>If this system property is set, statistics of the background indexing tasks are printed in the Carbon console.</p>	<p>This property can be used when you need to view the indexing statistics in the Carbon Console.</p>

sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableIndexThrottling=true		
s h <PRODUCT_HOME>/bin/wso2server.sh -DdisableMLSparkCtx=true	If this system property is set, the ML Spark context creation is disabled.	This property is used in scenarios where predictive analytics is performed in DAS. When WSO2 Machine Learner features are used in DAS, ML Spark context creation takes place by default. This prevents you from running the batch analytics features in DAS since it is not allowed to run multiple Spark contexts. Setting the -DdisableMLSparkCtx=true allows you to disable the ML Spark Context. After setting this property, you can continue using the predictive analytics features in DAS.
<p>This property is used only when an ML model is included in the DAS setup.</p>		

You can use multiple parameter values to disable multiple DAS components simultaneously. For example, you can use the following command to simultaneously disable both Spark server-related and Spark scripts execution-related components, which are described above.

```
sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableAnalyticsEngine=true -DdisableAnalyticsExecution=true
```

Purging Data

WSO2 DAS stores data in the [Data Access Layer \(DAL\)](#) and performs various analysis operations on them according to defined analytic queries. Thereby, as the volume of the data stored grows over time, the analysis and summarization jobs will also eventually consume more time. Then you can apply data purging to reduce the time taken to execute the analytics scripts, as well as the disk usage, since usually it is not necessary to analyze all data to produce the final result.

You can perform data purging in WSO2 DAS based on the following methods.

- [Per table data purging](#)
- [Global data purging](#)

Per table data purging

Follow the steps below to purge data of a selected table via the Management Console.

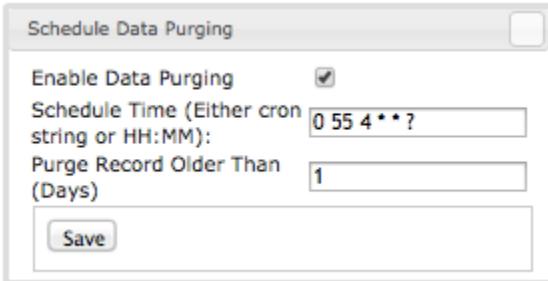
1. Log in to the management console as an admin user using the following URL: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Main**, and then click **Data Explorer**.
3. Select the required event table for the **Table Name** parameter, and click **Schedule Data Purging** as shown below to open the **Schedule Data Purging** dialog box.

The **Schedule Data Purging** option is displayed only for users of whom the assigned role has the **Delete** permission under **Record** enabled. For instructions on setting this permission to users, see [WSO2 DAS-Specific User Permissions](#).

Data Explorer

The screenshot shows the 'Data Explorer' page with a search bar at the top. Below it, there's a table configuration section with fields for 'Table Name*' (set to 'JMX_AGENT_TOOLBOX'), 'Maximum Result Count' (set to '1000'), and search options ('By Date Range', 'By Primary Key', 'By Query'). A red box highlights the 'Schedule Data Purging' button, which is located to the right of the table name field.

- Enter information as follows in the **Schedule Data Purging** dialog box to schedule the data purging task.



The fields of the above screen are described below.

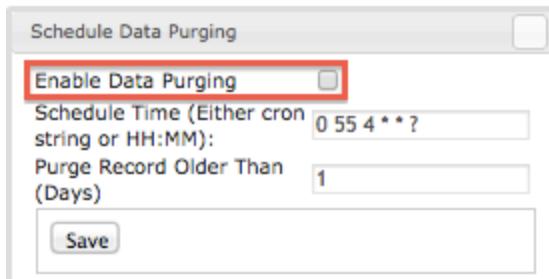
Field	Description
Enable Data Purging	Whether you want to enable data purging or not.
Schedule Time (Either cron string or HH:MM)	Enter the time on which you want to purge data via cron expression or by defining the time in the following format: HH:MM. For example, the following cron expression will configure the data purging job to run at 12:00 PM (noon) every day : 0 0 12 * * ?. For more information on cron expressions, go to Oracle Documentation .
Purge Record Older Than (Days)	Define the value as to keep data of only the last 'n' no of days back in the selected table. For example, if you give 1 as the value, the system will purge all data stored before yesterday.

- Click **Save**, and close the dialog box.

Removing scheduled data purging operations

Follow the steps below to remove a data purging operation that you have already scheduled.

- Log in to the Management Console as an admin user, if you are not already logged in.
- Click **Main**, and then click **Data Explorer**.
- Select the required table in the **Table Name** field, and then click **Schedule Data Purging** to open the **Schedule Data Purging** dialog box.
- Clear the **Enable Data Purging** check box as shown below.



5. Click **Save**, and close the dialog box.

Global data purging

You can perform data purging as a global operation which will affect all tenants. Follow the steps below to perform global data purging.

1. Navigate to <DAS_HOME>/repository/conf/analytics/analytics-config.xml file.
2. Change the configurations within the <analytics-data-purging> property as shown below.

```
<analytics-data-purging>
<!-- Below entry will indicate purging is enable or not. If user wants to enable
data purging for cluster then this property need to be enable in all nodes -->
<purging-enable>true</purging-enable>
<cron-expression>0 0 12 * * ?</cron-expression>
<!-- Tables that need include to purging. Use regex expression to specify the
table name that need include to purging.-->
<purge-include-table-patterns>
<table>.*</table>
<!--<table>.*jmx.*</table>-->
</purge-include-table-patterns>
<!-- All records that insert before the specified retention time will be eligible
to purge -->
<data-retention-days>365</data-retention-days>
</analytics-data-purging>
```

The properties of the above configuration file are shown below.

Property	Description
<purging-enable>	Change the value to true if you want to enable data purging.
<cron-expression>	The cron expression to define how you want to schedule the data purging operation. For example, the following cron expression will configure the archive job to run at 12:00 PM (noon) every day : 0 0 12 * * ?. For more information on cron expressions, go to Oracle Documentation .
<purge-include-table-patterns>	Specify the tables of which you want to purge data. By default, it is configured to perform data purging on all tables as follows: <table>.*</table> However, you can specify the required tables by defining a regular expression or a table name within the <table> property. Define one tag per each regular expression if you want to specify multiple tables.

<pre><data-retention-days></pre>	<p>Define the value as to keep data of only the last 'n' no of days back in the selected table. For example, the default value 365 will purge all data stored before 1 year.</p>
----------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Disabling data purging in a clustered mode

In a clustered mode, you can disable the scheduled data purging tasks being operated in a particular node (e.g. a node which is used for database tasks) using a startup parameter. Start the particular node by executing the following command in the CLI, to disable data purging in it: `sh <DAS_HOME>/bin/wso2server.sh -DdisableDataPurging=true`

Else, you can permanently use the startup parameter as a system property by adding the following line to the `<DAS_HOME>/bin/wso2server.sh` file.

Analytics Migration Tool

Instructions on how to migrate from WSO2 Business Activity Monitor 2.x to WSO2 DAS are as follows.

- Migrating analytics data
- Migrating WSO2 BAM toolboxes

Migrating analytics data

The analytics data migration tool can be used to migrate the Cassandra column family data of a WSO2 BAM 2.x server to WSO2 Data Analytics Server (DAS) 3.0.0. The `<DAS_HOME>/bin/ analytics-migrate.sh` script serves as the analytics migration tool.

Prerequisites

Before running the `<DAS_HOME>/bin/ analytics-migrate.sh` script, the event table(s) that serve as the destination for migrated data should be defined in WSO2 DAS. Follow the procedure below to create the required event tables by copying the relevant stream definitions from WSO2 BAM.

1. Log into the WSO2 BAM Management Console, and go to the **Main** tab.
2. Under **Registry**, click **Browse** to open the **Browse** page.
3. Navigate to `<Top_Folder>/system/governance/StreamDefinitions/<relevant_event_stream>/<relevant_stream_version>` as shown in the example below.

Browse

Root /

Location: /

Tree view Detail view

The screenshot shows the WSO2 Data Analytics Server Registry browser interface. The left pane displays a hierarchical tree view of registry nodes under the root. The tree includes categories like system, config, governance, event, permission, repository, and StreamDefinitions. Under StreamDefinitions, there are several entries: bam_mediation_stats_data_publisher, BAM_MESSAGE_TRACE, BAM_MESSAGE_TRACE_FILTER, HttpEventStream (with a red box around its '1.0.0' version), HttpEventStream2, and WSO2EventEventStream. Below StreamDefinitions are StreamIndexDefinitions and trunk, followed by local.

- /
- system
- config
- governance
- event
- permission
- repository
- StreamDefinitions
 - bam_mediation_stats_data_publisher
 - BAM_MESSAGE_TRACE
 - BAM_MESSAGE_TRACE_FILTER
 - HttpEventStream
 - 1.0.0
 - HttpEventStream2
 - WSO2EventEventStream
 - StreamIndexDefinitions
 - trunk
- local

4. Click on the stream version. This will open the **Detail View** tab for the event stream version you selected.
5. Click **Display as Text** to view the event stream configuration as a JSON array.

Home > Registry > Browse

Browse

Root /_system/governance/StreamDefinitions/HttpEventStream/1.0.0

Location:

Metadata

Properties

Content

```
{
  "streamId": "HttpEventStream:1.0.0",
  "name": "HttpEventStream",
  "version": "1.0.0",
  "nickName": "Event Stream",
  "description": "This is a test event stream",
  "metaData": [
    {
      "name": "server",
      "type": "STRING"
    }
  ],
  "payloadData": [
  ]
}
```

Permissions

6. Copy the JSON array to the clipboard.
7. Log into the DAS Management Console and go to the **Main** tab.
8. Click **Streams** to open the **Available Event Streams** page.
9. Click **Add Event Streams** to open the **Define New Event Stream** page.
10. Click **switch to source view**.
11. Clear the existing text in the source view and paste the JSON array you copied to the clipboard.
12. Click **Add Event Stream**.
13. If you want the event stream to persist events, follow the instructions in [Persisting Data for Batch Analytics](#).

Alternatively, you can redefine the complete event stream configuration as described in [Understanding Event Streams and Event Tables](#).

Sample command

```
./analytics-migrate.sh -cassandraUrl localhost -columnFamily
org_wso2_bam_phone_retail_store_kpi -analyticTable org_wso2_bam_phone_retail_store_kpi
-batchSize 1000 -tenantId -1234 -username admin -password admin -cassandraPort 9161
-clusterName Test Cluster
```

Parameters

The arguments that can be used with the command are described below.

Parameter	Description	Default Value	Example
-analyticTable	Destination name of the table which will have the migrated data.	None	testTable
-columnFamily	Name of the columnFamily to which analytics data will be migrated.	None	jmx_agent_toolbox
-tenantId	Tenant ID of the considered tenant.	super tenant	-1234
-serverUrl	The URL of the DAS server.	localhost	10.100.5.73
-serverPort	The Cassandra CQL port	9042	9042
-cassandraUrl	The URL to access the Cassandra server.	localhost	10.100.5.73
-cassandraPort	The thrift port for the Cassandra server.	9160	9160
-batchSize	Cassandra data is migrated from BAM to DAS in batches. This property specifies the size of a batch of data transferred at a given time.	1000	1000
-username	The username to access the Cassandra server.	None	admin
-passwords	The password to access the Cassandra server.	None	admin
-clusterName	The name of the cluster to which the data should be migrated.	None	cluster001

Migrating WSO2 BAM toolboxes

Toolboxes concept of [WSO2 Business Activity Monitor](#) is replaced in WSO2 DAS by a CAR file based deployment approach. Since you need to map Hive queries to the Apache Spark syntax that is supported by WSO2 DAS, you cannot directly upload WSO2 BAM (2.x versions) toolboxes to WSO2 DAS. Thereby, you need to wrap all the artifacts (stream definitions, event receivers, event stores, event publishers, Spark scripts etc.) in a .car file and upload it to WSO2 DAS. For instructions on how to create a Carbon application with the CAR extension, see [Packaging Artifacts as a C-app Archive](#). However, you can use the [analytics migration tool](#) to migrate the data from Cassandra to the DAS datasource.

Analytics Data Backup / Restore Tool

The analytics data backup / restore tool can be used to back up the already existing record store and file system of a DAS server to a specific directory in our machine. The backed up data can be restored later from the same directory to the current DAS node.

The following script is used to backup and restore data.

- On Windows: <PRODUCT_HOME>\bin\analytics-backup.bat --run
- On Linux/Solaris/Mac OS: <PRODUCT_HOME>/bin/analytics-backup.sh

The following table describes the arguments that are used with the backup script mentioned above.

Argument	Purpose
----------	---------

-backupRecordStore	To backup the record store wherethepersisted events are saved.
-dir <directory>	Directoryused as the target when backing up the record store/file system, or a already backed up record store/file system. Thedirectlyshould be specified in the backup or restore data.
-restoreRecordStore	To restore a record store. Before you use this command, the record store you already backed up.
-deleteTables	<p>Deletes the specified table list. You need to specify the table list to be deleted later.</p> <div style="border: 1px solid black; padding: 10px;"> <p>Once you perform a deleteTable action, you need to send a <code>DELETE</code> <code>AS-URL> : <PORT> /analytics/tables/<TABLE-NAME>/schema</code> to remove the in-memory schema.</p> </div>
-purge	Allows data purging for a given time range.
-enableIndexing	<p>Indexing is disabled in the data restoration step by default. This is because if the target DAS server is already running, that server will also index the same data causing an index conflict. If the target DAS server is not running, you can index the data at the time the target server is started up is indexed in the usual way.</p>

-reindexEvents	This switch will make the tool re-index the data already there in a table. This can be used where an indexing is corrupted for some reason, or if some of the older data was not mentioned in the schema etc..
-migrateTableSchemaV30To31	This option migrates the analytics tables metadata from v3.0.x to v3.1.0+.
-tables <table list>	<p>This argument is used to specify the list of event tables that should be backed up.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>If you do not use this argument to back up data from one or more specific event tables, all event tables will be backed up.</p> </div>
-tenantId <tenant id (default is super tenant)>	This argument is used to select the tenant ID(s) of which the events should be backed up.
-timeFrom <yy-mm-dd-hh:mm:ss>	<p>This argument specifies the starting time (inclusive) when defining the time range considered when backing up/restoring events.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>This argument should be used together with the -timeto argument.</p> </div>

- timeTo
<yy-mm-dd-hh:mm:ss>

This argument specifies the ending time (inclusive) when defining the time considered when backing up/restoring events.

This argument should be used together with the -timefrom argument.

All the arguments described in the above table are displayed in your console when you run the <PRODUCT_HOME>\bin\analytics-backup.bat --run or the <PRODUCT_HOME>/bin/analytics-backup.sh without adding any argument.

Sample commands

Backing up data

```
./analytics-backup.sh -backupRecordStore -tables Table1,Table2,Table3 -dir /home/user/backup
```

The above command backs up the records of the Table1, Table2, Table3 event tables as well as the related indexing information in the /home/user/backup directory. Only the data entered between 8.00 AM on 11th August 2015 and 10.30 PM on 15th October 2015 are selected to be backed up.

Restoring data

```
./analytics-backup.sh -restoreRecordStore -dir /home/user/backup
```

The above command restores records of the Table1, Table2, Table3 event tables as well as the related indexing information from the /home/user/backup directory. Only data entered between 9.00 - 10.00 AM on 11th August 2015 are selected to be restored.

Performance Tuning

This section describes some recommended performance tuning configurations to optimize the performance of WSO2 DAS. It assumes that you have set up WSO2 DAS 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
- DAS-Level settings
 - Receiving events
 - Publishing events
 - Spark Cluster Tuning

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.

```
net.ipv4.tcp_fin_timeout = 30
fs.file-max = 2097152
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.core.rmem_default = 524288
net.core.wmem_default = 524288
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.ipv4.ip_local_port_range = 1024 65535
```

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 `<DAS_HOME>/bin/wso2server.bat` file (for Windows) or the `<DAS_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.

```
<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](#).

Property	Description	Recommendation
<code>maxActive</code>	The maximum number of active connections that can be allocated from the connection pool at the same time. The default value is 100.	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.
<code>minIdle</code>	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.	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.

testOnBorrow	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.	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.
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.

DAS-Level settings

Performance tuning can be tried out in the following areas at the DAS level. The performance is considered in terms of throughput per second (TPS) and latency.

- Configuration patterns

Receiving events

The following parameters which affect the performance relating to receiving events are configured in the <DAS_HOME>/repository/conf/data-bridge/data-bridge-config.xml file. These configurations are common for both thrift and binary protocols.

Property	Description	Default Value	Recommendation
----------	-------------	---------------	----------------

workerThreads	The number of threads reserved to handle the load of events received.	10	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.
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 .	10	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.
eventBufferSize	The number of messages that is allowed in the receiving queue at a given time.	2000	This value should be increased when there is an increase in the receiving throughput .

Publishing events

The following parameters which affect the performance relating to publishing events are configured in the `<Das_Home>/repository/conf/data-bridge/data-agent-config.xml` file. These configurations are common for both thrift and binary protocols.

Property	Description	Default Value	Recommendation
QueueSize	The size of the queue event disruptor which handles events before they are published to an application/data store.	32768	<p>The value specified should always be the result of an exponent with 2 as the base. (e.g., 32768 is 2^{15}).</p> <p>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.	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.

CorePoolSize	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.	1	The number of available CPU cores should be taken into account when specifying this value. Increasing the core pool size may improve the throughput, but latency will also be increased due to context switching.
MaxPoolSize	The maximum number of threads that should be reserved at any given time to handle events.	1	The number of available CPU cores should be taken into account when specifying this value. Increasing the maximum core pool size may improve the throughput since more threads can be spawned to handle an increased number of events. However, latency will also increase since a higher number of threads would cause context switching to take place more frequently.

For better throughput you can configure the parameters as follows.

```
<QueueSize>32768</QueueSize>
<BatchSize>200</BatchSize>
<CorePoolSize>1</CorePoolSize>
<MaxPoolSize>1</MaxPoolSize>
```

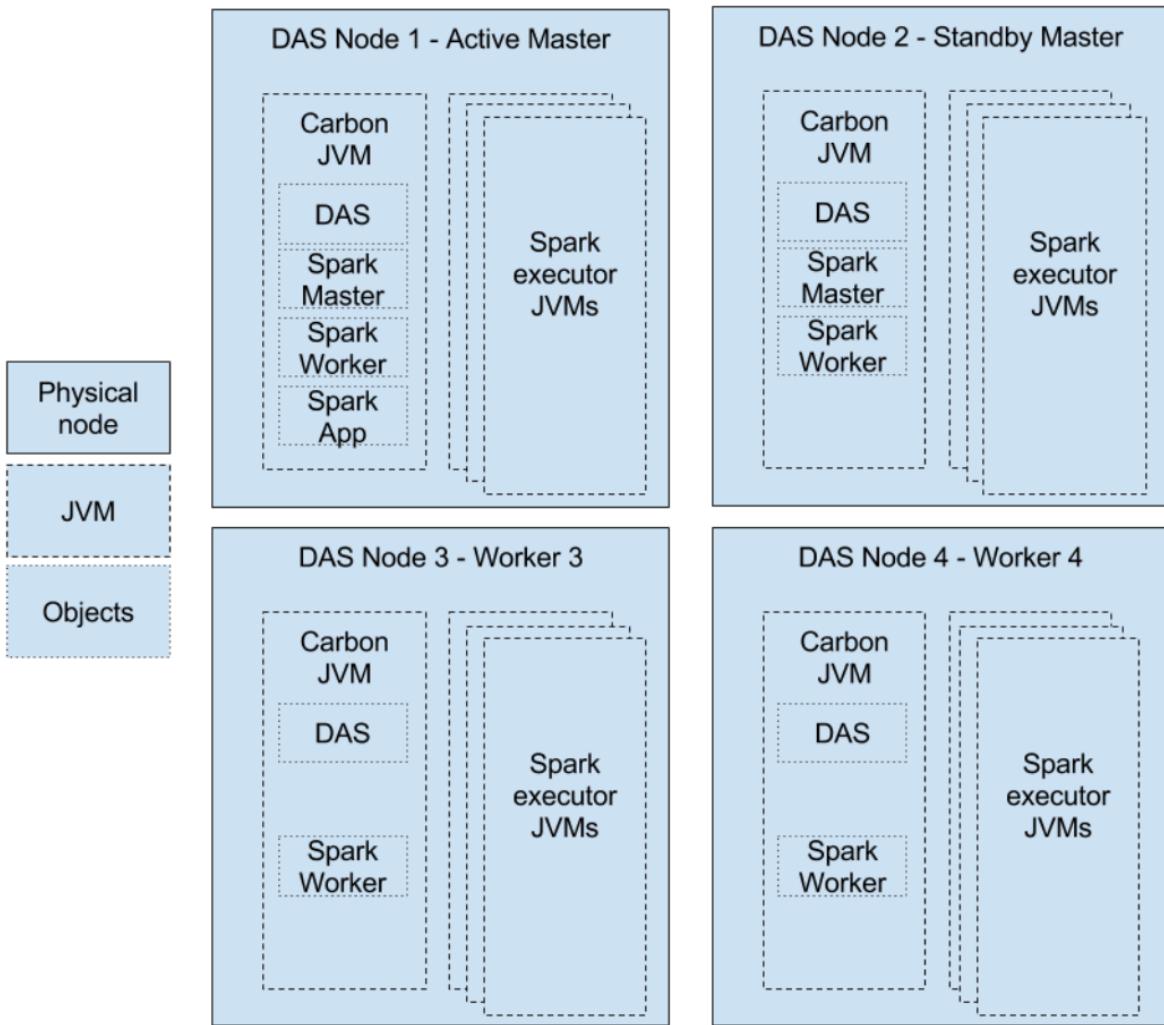
For reduced latency, you can configure the parameters as follows.

```
<QueueSize>256</QueueSize>
<BatchSize>200</BatchSize>
<CorePoolSize>1</CorePoolSize>
<MaxPoolSize>1</MaxPoolSize>
```

Spark Cluster Tuning

In a DAS production environment, it is important to allocate the resources correctly to each node, in order to achieve optimum performance.

The following diagram depicts a typical DAS multimode set up.



The resource allocation to nodes should be carried out depending on the requirement based on the function on each node. The resource allocation is specified by configuring the `<Das home>/repository/conf/analytics/spark/spark-defaults.conf` file. This file contains the following two categories of configurations.

- **Carbon related configurations:** These are carbon specific properties that are applicable when running Spark in a WSO2 Carbon environment and they start with the `carbon` prefix.
- **Spark related configurations:** These are the default properties shipped with Apache Spark.

Parameters to be configured are as follows.

Cores

Parameter	Default Value	Description
<code>spark.executor.cores</code>	All the available cores on the worker.	The number of cores to use on each executor. Setting this parameter allows an application to run multiple executors on the same worker, provided that there are enough cores on that worker. Otherwise, only one executor per application is run on each worker.
<code>spark.cores.max</code>	<code>Int.MAX_VALUE</code>	The maximum amount of CPU cores to request for the application from across the cluster (not from each machine).
<code>spark.worker.cores</code>	1	The number of cores assigned for a worker.

Memory

Parameter	Default Value	Description
spark.worker.memory	1g	Amount of memory to use per worker, in the same format as JVM memory strings (e.g., 512m, 2g).
spark.executor.memory	512m	Amount of memory to use per executor process, in the same format as JVM memory strings (e.g., 512m, 2g).

The number of executors in a single worker for the carbon-application can be derived as follows:

```
number of executors in a single worker = FLOOR
( MIN (spark.worker.cores, spark.cores.max) / spark.executor.cores )
```

Then the amount of memory which should be allocated for a worker should be:

```
spark.worker.memory = spark.executor.memory × number of executors
```

Configuration patterns

By setting different values for each of the parameters above, we can have different configuration patterns.

You can consider an **AWS m4.xlarge instance** for an example. It has 8 vCPUs and 16 GB memory. If you allocate 4 GB and 4 cores to the OS and the Carbon JVM (by default this only takes 1GB memory), then you can allocate `spark.worker.memory = 12g` and `spark.worker.cores = 4`.

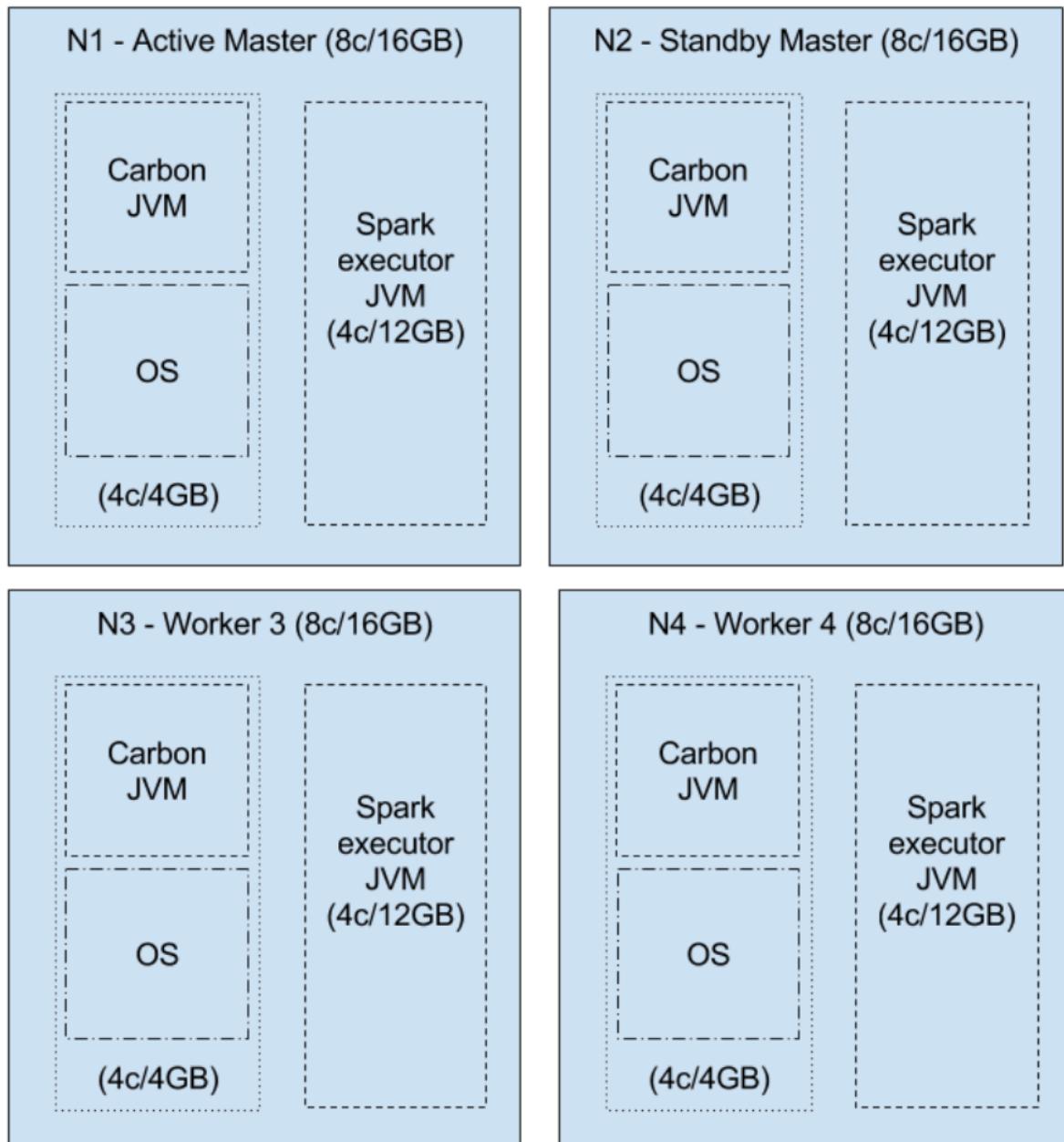
Single executor workers

If you do not specify a value for the `spark.cores.max` or `spark.executor.cores` property, then all the available cores is taken up by one executor.

```
Executors = min (4,Int.MAX_VALUE)/4 = 1
```

Therefore, all the memory for that executor can be allocated as follows.

```
spark.executor.memory=12g
```



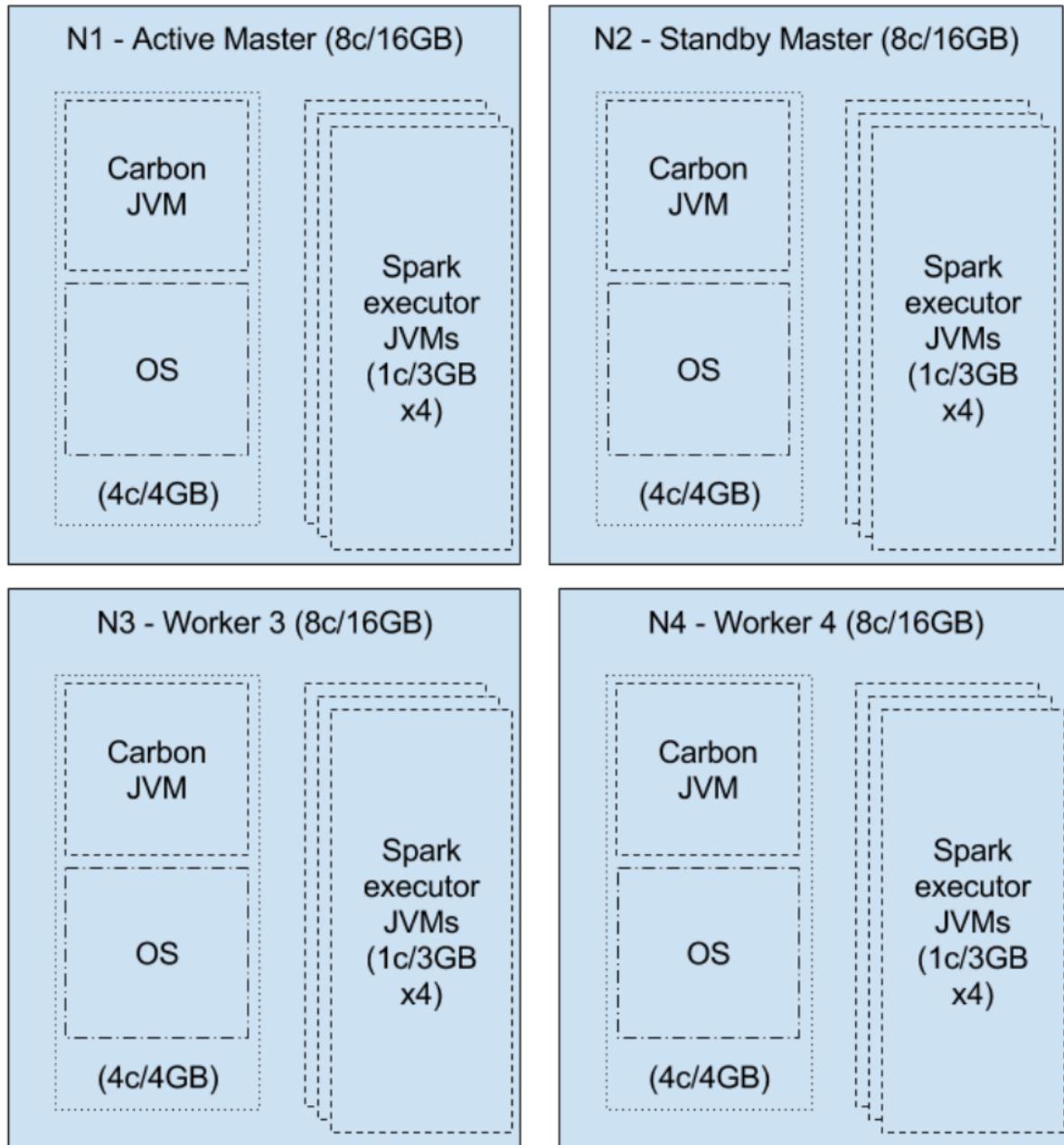
Having large amount of memory for a single JVM is not advisable, due to GC (Garbage Collection) performance.

Multiple executor workers

If `spark.executor.cores` property is set to 1, then the number of executors can be derived as follows:

```
min (4, Int.MAX_VALUE) / 1 = 4
```

Therefore, $12\text{GB}/4 = 3\text{GB}$ can be allocated per executor.

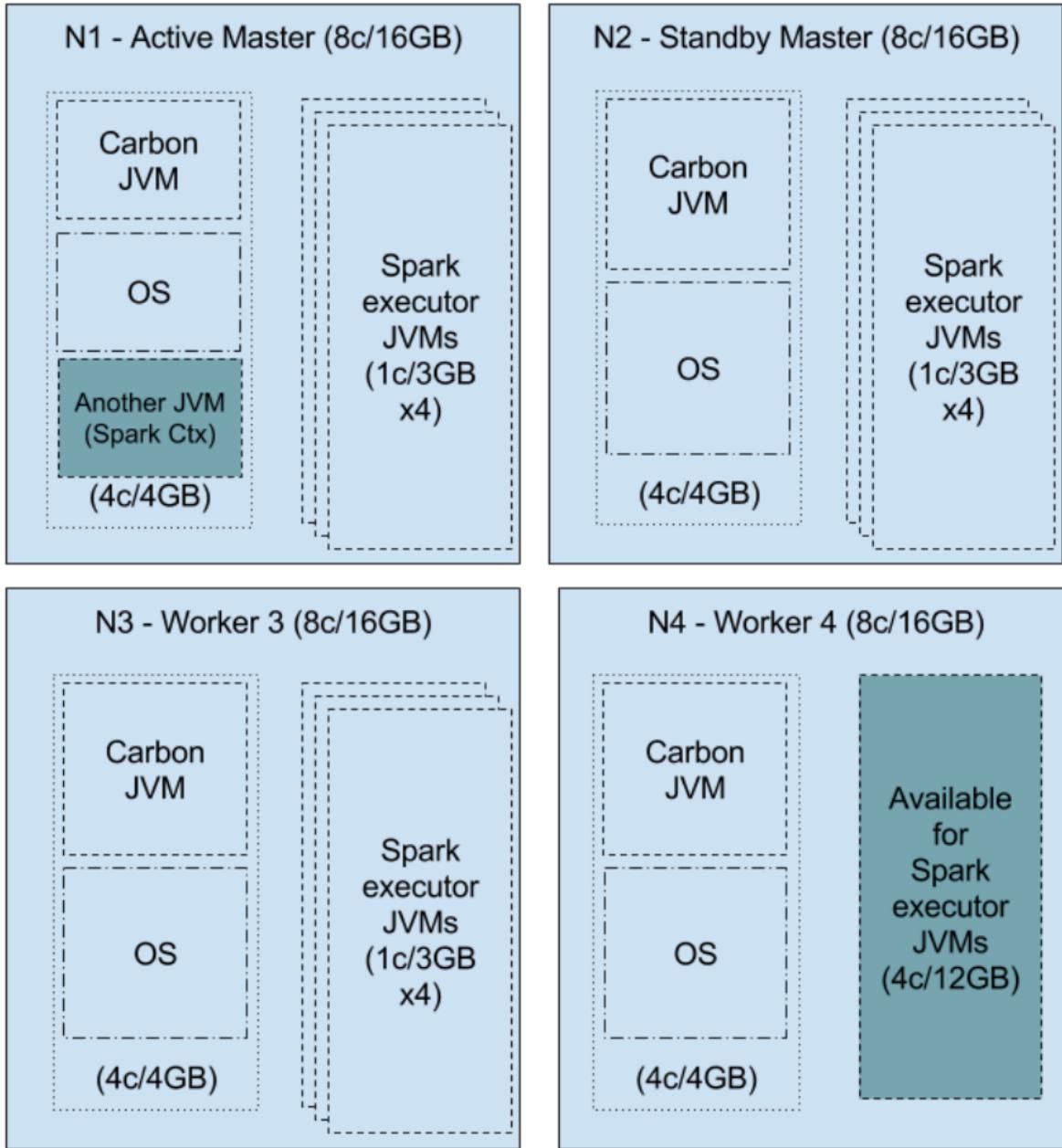


Resource limited workers

The number of cores used for the carbon application cluster-wide can be limited by specifying a value for the `spark.cores.max` property.

If `spark.cores.max = 3` per node \times 4 nodes = 12, there are 4 excess cores that can be used by some other application.

Let us consider the above multiple executor setup.



Here, there are resources for 16 executors with 16 cores and 48GB of memory. With the `spark.cores.max = 12` (i.e. 3×4), 12 executors are assigned to the carbon application and the rest of the cores and memory can be assigned to another Spark application (i.e. 4 cores and 12 GB are available in the cluster and that can be used by the application, depending on its preference).

Configuration Guide

The following topics explore different configuration options to customize the product according to common user-specific scenarios.

- Registry
- User Management
- Feature Management
- Logging
- Datasources
- Server Roles for C-Apps
- Scheduling Tasks
- Security

[Go to Home Page](#)

Registry

This chapter contains the following information:

- [Introduction to Registry](#)
- [Managing the Registry](#)
- [Searching the Registry](#)
- [Sharing Registry Space Among Multiple Products](#)

Note

The screenshots in this section may vary depending on the product and the configuration options you are using.

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.

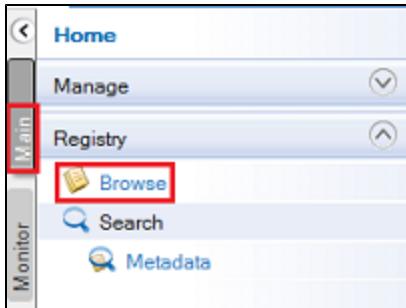
- **Local Data Repository** - Used to store settings/metadata specific to the product. This registry is not intended to be shared among multiple servers.
- **Configuration Registry** - Used to store product-specific configurations. These configurations can be shared across multiple instances of the same 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.

Entries

Name	Created On	Author
config	06 Oct	wso2.system..
governance	06 Oct	wso2.system..
local	06 Oct	wso2.system..

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,

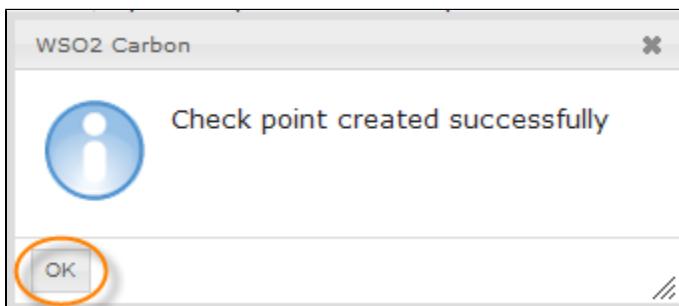
Metadata	
	 Feed
Created:	By wso2.system.user 13 May 12:05:58
Last Updated:	By wso2.system.user 27 May 12:05:39
Media Type:	Unknown
Checkpoint:	 Create Checkpoint
Versions:	 View versions
Description:	Governance registry of the carbon server. This collection is used to store the resources common to the whole platform.  Edit

Checkpoint Creation

1. To create a checkpoint, click on the *Create Checkpoint* link.

Metadata	
	 Feed
Created:	By wso2.system.user 20 May 14:05:26
Last Updated:	By wso2.system.user 20 May 14:05:26
Media Type:	application/vnd+wso2.sequence
Checkpoint:	 Create Checkpoint
Versions:	 View versions
Description:	 Edit

2. If the checkpoint was successfully created, a message will be displayed. Click *OK*.



Viewing Versions

1. Click on the *View versions* link.

Metadata	
 Feed	
Created:	By wso2.system.user 20 May 14:05:26
Last Updated:	By wso2.system.user 20 May 14:05:26
Media Type:	application/vnd+wso2.sequence
Checkpoint:	 Create Checkpoint
Versions:	 View versions
Description:	 Edit

2. The *Versions* page appears.

Versions of /_system/config/Sequence_2			
Version	Last Modified Date	Last Modified By	Actions
318	20 May 14:05:26	wso2.system.user	 Details Restore
117	20 May 12:05:13	wso2.system.user	 Details Restore

- 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.

Description: <input type="text" value="Configuration registry of the carbon server. This collection is used to store the resources of this product cluster."/>	Configuration registry of the carbon server. This collection is used to store the resources of this product cluster. <input type="button" value="Save"/> <input type="button" value="Cancel"/>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- Restore - Allows to restore a resource/collection version.

Editing Description

1. To edit a description of a resource/collection, click on the *Edit* link.

Metadata

Feed

Created:	By wso2.system.user 13 May 12:05:58
Last Updated:	By wso2.system.user 20 May 14:05:26
Media Type:	Unknown
Checkpoint:	Create Checkpoint
Versions:	View versions
Description:	Configuration registry of the carbon server. This collection is used to store the resources of this product cluster. Edit

2. Edit the description of a resource/collection in the text area and click Save.

Description:	Configuration registry of the carbon server. This collection is used to store the resources of this product cluster.
Save Cancel	

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](#).

Properties

Add New Property

1 Property

Name	Value	Action
prop1	val	Edit Delete

Adding a Property

1. To add a property, click on the *Add New Property* link.

Properties

Add New Property

1 Property

Name	Value	Action
prop1	val	Edit Delete

2. In the *Add New Property* panel, enter a unique name of a property and its value. Click *Add*.

The screenshot shows the 'Properties' dialog box. In the top left, there's a green plus icon labeled 'Add New Property'. Below it, a form titled 'Add New Property' has two fields: 'Name*' containing 'prop2' and 'Value*' containing 'value1'. Both fields have orange outlines. At the bottom of this form are 'Add' and 'Cancel' buttons. Below this form is a table titled '1 Property' with one row. The row contains 'Name' 'Value' and 'Action' columns. The 'Name' column has 'prop1', the 'Value' column has 'val', and the 'Action' column has 'Edit' and 'Delete' buttons, also with orange outlines.

Name	Value	Action
prop1	val	Edit Delete

Editing a Property

1. Click on the *Edit* link of a particular property in the *Action* column.

The screenshot shows the 'Properties' dialog box with '2 Properties' listed. The table has three rows: one for 'prop1' with value 'val', and two for 'prop2' with value 'value1'. The 'Action' column for each row contains 'Edit' and 'Delete' links. The 'Edit' link for the first row ('prop1') is highlighted with an orange circle.

Name	Value	Action
prop1	val	Edit Delete
prop2	value1	Edit Delete

2. Edit the name and the value of a property in the active fields and click *Save*.

The screenshot shows the 'Properties' dialog box with '2 Properties' listed. The table has three rows. The 'Name' and 'Value' fields for the first row ('prop1') are outlined with orange circles. The 'Action' column for this row contains 'Save' and 'Cancel' buttons, also with orange outlines.

Name	Value	Action
prop1	val	Save Cancel
prop2	value1	Edit Delete

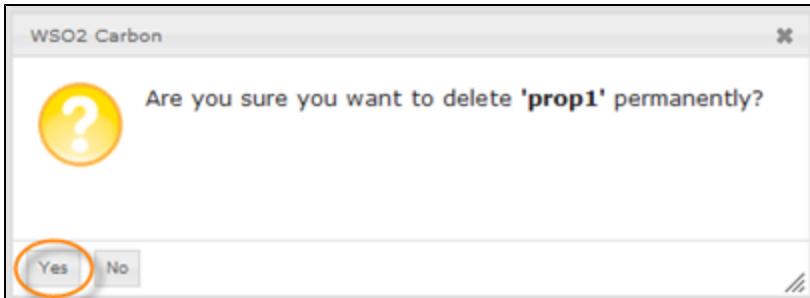
Deleting a Property

1. To delete a property, click on the *Delete* link of a certain property in the *Action* column.

The screenshot shows the 'Properties' dialog box with '2 Properties' listed. The table has three rows. The 'Action' column for the first row ('prop1') contains 'Edit' and 'Delete' links. The 'Delete' link for the first row is highlighted with an orange circle.

Name	Value	Action
prop1	val-new	Edit Delete
prop2	value1	Edit Delete

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.

Name	Created On	Author
_system	28 Sep	wso2.system..

- 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

Name	Created On	Author
config	28 Sep	wso2.system..

To see the available actions over a resource, click on the *Actions* icon.

Name	Created On	Author
config	28 Sep	wso2.system..

The following actions over the resources are available:

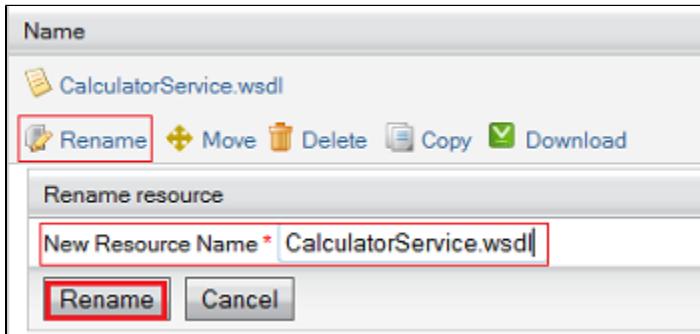
- **Rename** - Allows to rename a resource.
- **Move** - Allows to move a resource to a new directory.
- **Delete** - Allows to delete a resource.
- **Copy** - Allows to copy a resource to a specified directory.

Tip

All these options are available not for all the resources.

Renaming a Resource

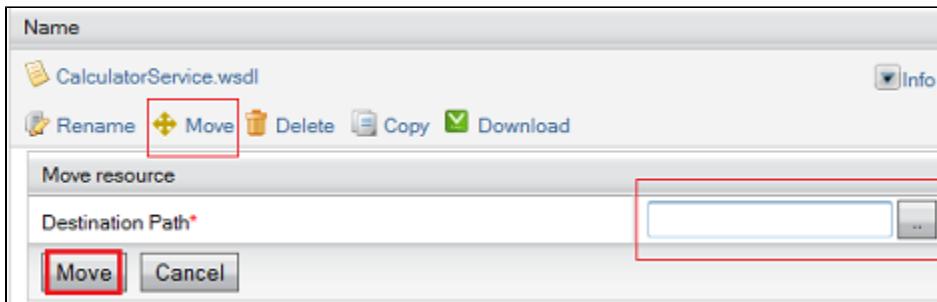
1. To rename a resource, click *Rename* and enter a new name to the field.



2. Click on the *Rename* button to save a new name of a resource.

Moving a Resource

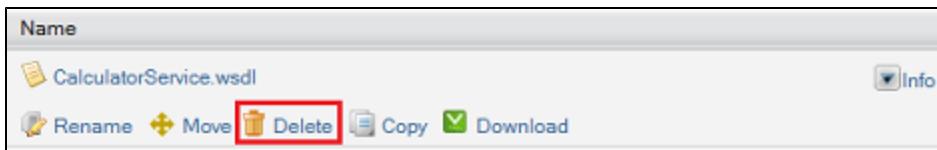
1. To move a resource to a new directory, click *Move* and specify *Destination Path*.



2. Click *Move*.

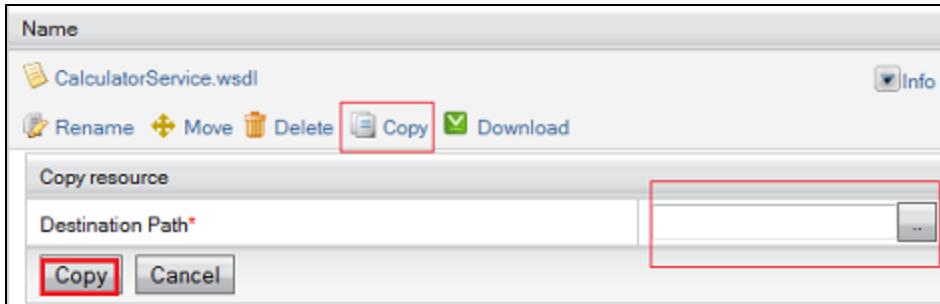
Deleting a Resource

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*.



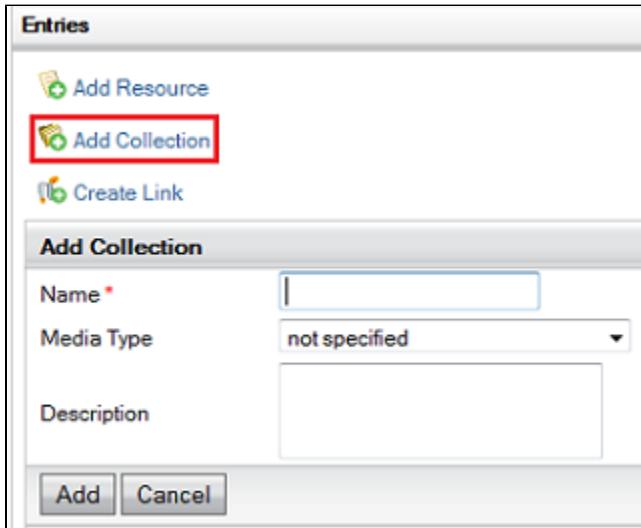
2. Click *Copy*.

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
- **Description** - A brief description of a collection.

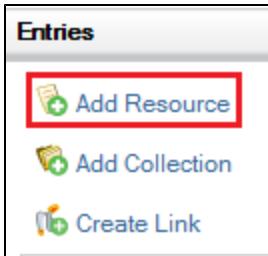
3. Click *Add*.

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.



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**

This image shows a screenshot of a dialog box titled 'Uploading Content from File'. At the top, there is a 'Method' dropdown menu with four options: 'Upload content from file', 'Import content from URL', 'Create Text content', and 'Create custom content'. The 'Upload content from file' option is highlighted with a blue selection bar and has a red border around it. Below the dropdown is a 'File*' input field with the path 'C:\wso2\Chad.wsdl' entered. A 'Browse...' button is next to the input field. A note below says 'Give the path of a file to fetch content (xml,wsdl,jar etc.)'.

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.

This image shows a screenshot of the 'Add Resource' dialog box. The 'Method' dropdown at the top is set to 'Upload content from file'. The main area is titled 'Upload Content From File' and contains four form fields:

- File ***: Input field containing 'C:\wso2\Chad.wsdl' with a 'Browse...' button to its right. A note below says 'Give the path of a file to fetch content (xml,wsdl,jar etc.)'.
- Name ***: Input field containing 'Chad.wsdl'.
- Media type**: Input field containing 'application/wsdl+xml'.
- Description**: Text area containing 'test'.

 At the bottom are two buttons: 'Add' (highlighted with a red box) and 'Cancel'.

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.

Add Resource

Method	Import content from URL
Import Content From URL	
URL *	http://localhost:8282/services/Calculator?wsdl2
Name *	Calculator.wsdl2
Media type	
Description	
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

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.

Add Resource

Method	Create Text content
Create Text Content	
Name *	TestResource
Media type	text/plain
Description	
Content	<input checked="" type="radio"/> Plain Text Editor <input type="radio"/> Rich Text Editor <pre><?xml version="1.0" encoding="UTF-8"?> <!-- Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF 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,</pre>
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

Custom Content Creation

1. If this method was selected, choose the *Media Type* from the drop-down menu and click *Create Content*.

The screenshot shows the 'Add Resource' dialog box. At the top, there's a dropdown labeled 'Method' set to 'Create custom content'. Below it is a section titled 'Create Custom Content' with a note: 'Select the corresponding media type from the list below.' A dropdown menu labeled 'Media type' is open, showing 'application/vnd.wso2-profiles+xml'. There are two buttons at the bottom: 'Create Content' and 'Cancel'.

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.

The screenshot shows the 'Entries' panel with three options: 'Add Resource', 'Add Collection', and 'Create Link'. The 'Create Link' option is highlighted with a red box. Below it is the 'Create Link' dialog box. It has a 'Method' dropdown set to 'Add Symbolic Link'. Underneath is a 'Add Symbolic Link' form with two required fields: 'Name*' and 'Path*', both with red asterisks. At the bottom are 'Add' and 'Cancel' buttons.

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.

Create Link

Method **Add Symbolic Link**

Add Symbolic Link

Name *	symbol1
Path*	/_system

Add **Cancel**

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.

Create Link

Method **Add Remote Link**

Add Remote Link

Name *	remote1
Remote Instance*	No instances available
Path	

Add **Cancel**

Managing Role Permissions

The **Permissions** panel shows the defined role permissions, allows to [add](#) new role permissions and [edit](#) existing ones.

Permissions

New Role Permissions

Role	wso2.anonymous.role	Action	-Select-	<input checked="" type="radio"/> Allow <input type="radio"/> Deny
Add Permission				

Defined Role Permissions

Role	Read		Write		Delete		Authorize	
	Allow	Deny	Allow	Deny	Allow	Deny	Allow	Deny
everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>						

Apply All Permissions **Reset**

Adding New Role Permissions

1. In the *New Role Permission* pane, select a role to set a permission.

Permissions

New Role Permissions

Role	wso2.anonymous.role	Action	-Select-	<input checked="" type="radio"/> Allow <input type="radio"/> Deny
Add Permission				

2. Select an action from the drop-down menu. The following actions are available:

- **Read**
- **Write**
- **Delete**
- **Authorize**

Permissions

New Role Permissions

Role	everyone	Action	Read	<input checked="" type="radio"/> Allow <input type="radio"/> Deny
Add Permission				

3. Select whether to allow the action the selected role or deny.

Permissions

New Role Permissions

Role	everyone	Action	Read	<input checked="" type="radio"/> Allow <input type="radio"/> Deny
Add Permission				

4. Click on the *Add Permission* button.

Permissions

New Role Permissions

Role	everyone	Action	Read	<input checked="" type="radio"/> Allow <input type="radio"/> Deny
Add Permission				

5. A new permission appears in the *Defined Role Permissions* list.

Role	Read		Write		Delete		Authorize	
	Allow	Deny	Allow	Deny	Allow	Deny	Allow	Deny
everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Apply All Permissions Reset								

Editing Role Permissions

1. You can also edit the defined role permissions using the check boxes in the *Defined Role Permissions* list.

Role	Read		Write		Delete		Authorize	
	Allow	Deny	Allow	Deny	Allow	Deny	Allow	Deny
everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Apply All Permissions				Reset				

2. After editing the permissions, click on the *Apply All Permissions* button to save the alterations.

Searching the Registry

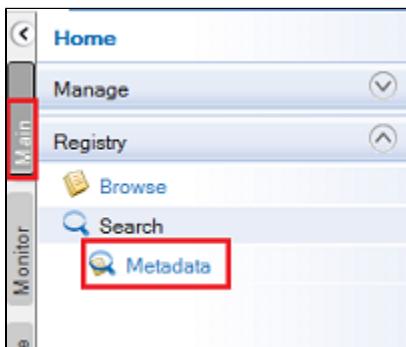
All resources found in the Registry can be searched through the product's Management Console. Search can be refined by resource name, created date range, updated date range, tags, comments, property name, property value, media type etc.

Tip

To search for matches containing a specific pattern, use the "%" symbol.

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.

Search

Search Registry

Load a Saved Search
Fill the Search Options from a previous search.

Filter Name

Search Options

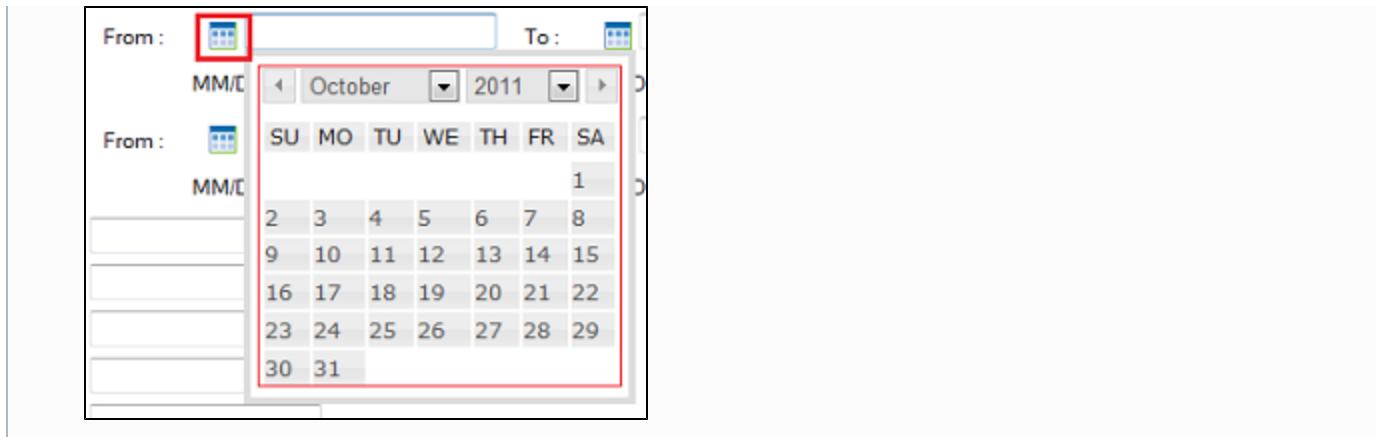
Resource Name	<input type="text"/>
Created	From : <input type="button" value="Calendar"/> <input type="text"/> To : <input type="button" value="Calendar"/> <input type="text"/> MM/DD/YYYY MM/DD/YYYY
Updated	From : <input type="button" value="Calendar"/> <input type="text"/> To : <input type="button" value="Calendar"/> <input type="text"/> MM/DD/YYYY MM/DD/YYYY
Created by	<input type="text"/>
Updated by	<input type="text"/>
Tags	<input type="text"/>
Comments	<input type="text"/>
Property Name	<input type="text"/>
Property Value	<input type="text"/>
Media Type	<input type="text"/>

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 author of a resource updating.
- **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

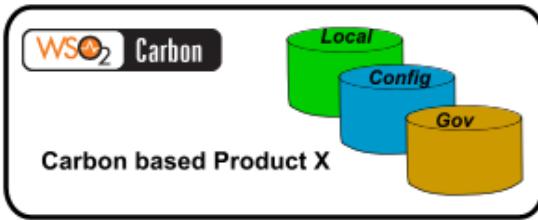
Pattern #1 : Local Registry

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.

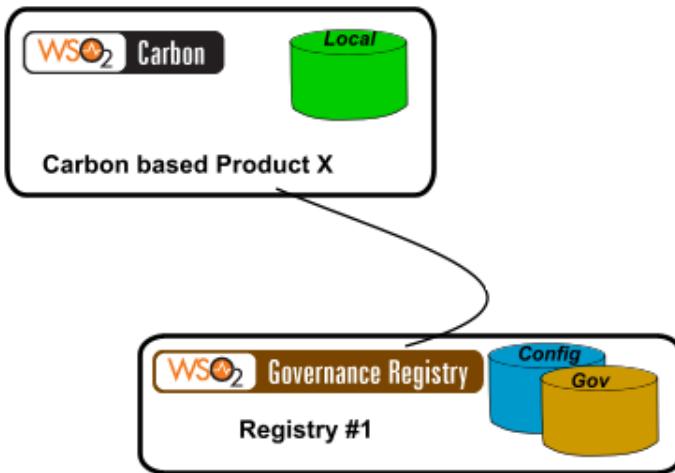
Pattern #2 : Remote Conf, Go.Reg

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.

1. Download and extract WSO2 Governance Registry distribution from <http://wso2.com/products/governance-registry>.
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:

```
<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.

4. Copy the 'mySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to G-REG_HOME/repository/components/lib directory.

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.
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.

```

<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>

```

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>regadmin@jdbc:mysql://10.20.30.41:3306/registrydb</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:

```
<clustering class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
enable="true" />
```

Note

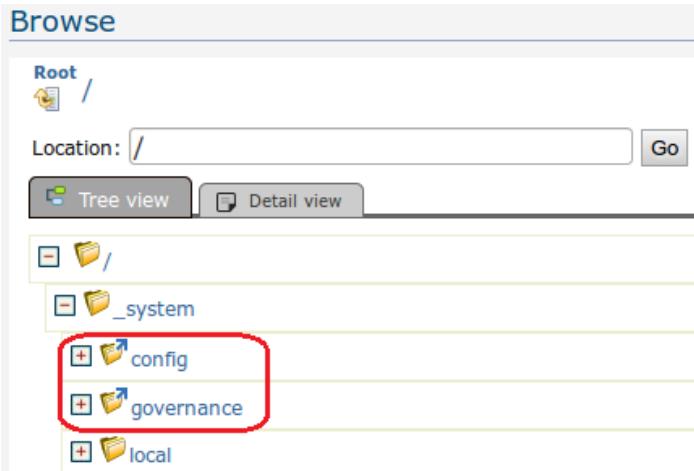
The above configuration is needed only when caching is enabled in the server instances and <enableCache> parameter set to true. Clustering enables cache invalidation in configured nodes at the event of any changes to the registry resources by any of the Carbon server nodes in the deployment setup.

2. Copy 'MySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to \${G-REG_HOME}/repository/components/lib in both Carbon server instances.

3. Start both servers and note the log entries that indicate successful mounting to the remote Governance Registry instance. For example,

```
[2012-12-14 14:02:31,309] INFO - CarbonCoreActivator Java H   : /home/gillian/Products/Mount/Node1
[2012-12-14 14:02:31,309] INFO - CarbonCoreActivator Java Temp Dir : /home/gillian/Products/Mount/Node1/tmp
[2012-12-14 14:02:31,310] INFO - CarbonCoreActivator User       : gillian, en-US, Asia/Colombo
[2012-12-14 14:02:31,383] INFO - AgentDS Successfully deployed Agent Client
[2012-12-14 14:02:35,145] INFO - EmbeddedRegistryService Configured Registry in 54ms
[2012-12-14 14:02:35,182] INFO - EmbeddedRegistryService Connected to mount at remote_registry in 4ms
[2012-12-14 14:02:35,460] INFO - EmbeddedRegistryService Connected to mount at remote_registry in 1ms
[2012-12-14 14:02:35,495] INFO - RegistryCoreServiceComponent Registry Mode : READ-WRITE
[2012-12-14 14:02:38,739] INFO - ClusterBuilder Clustering has been disabled
[2012-12-14 14:02:39,646] INFO - LandingPageWebappDeployer Deployed product landing page webapp: StandardEngine[Catalina].st
st].StandardContext[/home]
[2012-12-14 14:02:39,729] INFO - HttpCoreNIOSSLSender Loading Identity Keystore from : repository/resources/security/wso2car
[2012-12-14 14:02:39,783] INFO - HttpCoreNIOSSLSender Loading Trust Keystore from : repository/resources/security/client-tru
```

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.

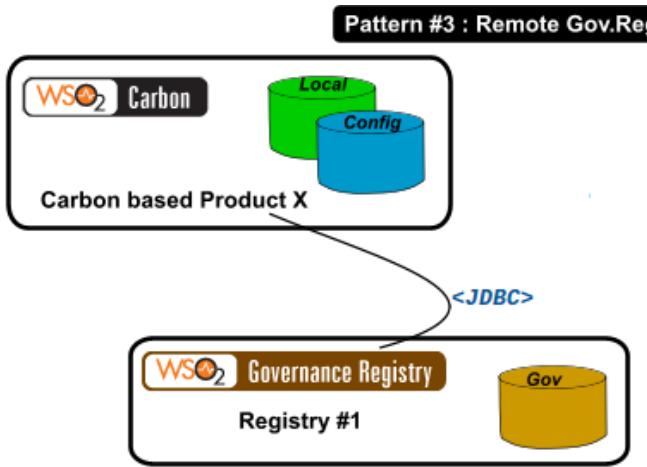


Figure 3: Governance partition in the remote Governance Registry instance.

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.

1. Download and extract WSO2 Governance Registry distribution from <http://wso2.com/products/governance-registry>.
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:

```
<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 clustering with the following configuration.

```
<clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
enable="false"/>
```

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.

4. Copy the 'mySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to G-REG_HOME/repository/components/lib directory.
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.

```

<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>

```

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>regadmin@jdbc:mysql://10.20.30.41:3306/registrydb</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.

```
<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.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
enable="false"/>
```

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.

6. Copy 'MySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to \${G-REG_HOME}/repository/components/lib in both Carbon server instances.

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.

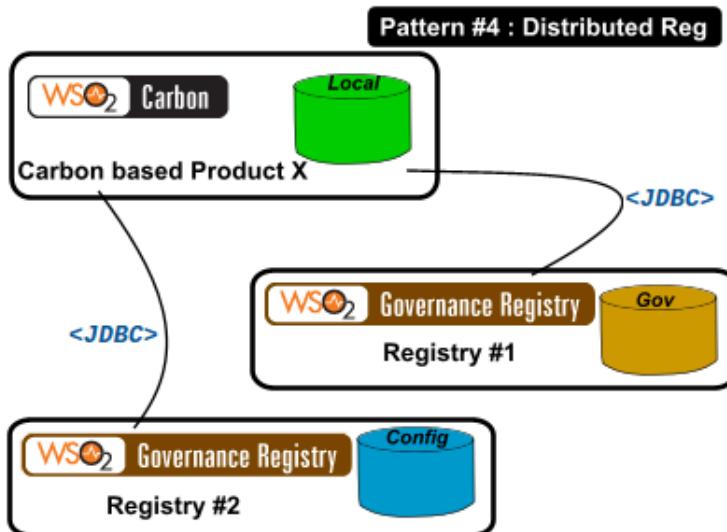


Figure 4: Config and governance partitions in separate registry instances.

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
```

2. Enter the password when prompted.

3. 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.

1. Download and extract WSO2 Governance Registry distribution from <http://wso2.com/products/governance-registry>.

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:

```
<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://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>
```

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:

```
<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://10.20.30.42:3306/registrydb2</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>
```

4. Repeat the same for G-Reg 3 as follows.

```

<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://10.20.30.43:3306/registrydb3</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>

```

5. Navigate to \$G-REG_HOME/repository/conf/axis2/axis2.xml file in all instances and enable clustering with the following configuration.

```

<clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
enable="false"/>

```

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.

6. Copy the 'mySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to 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 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 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 that share the remote registry instances. Namely, Foo product cluster shares G-Reg 1 and G-Reg 2 while Bar 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 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>
            <url>jdbc:mysql://10.20.30.42:3306/registrydb2</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 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,

```

<dbConfig name="governance_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
<dbConfig name="config_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG_CONFIG</dataSource>
</dbConfig>

```

Specify the remote Governance Registry instance with the following configuration:

```

<remoteInstance url="https://10.20.30.41:9443/registry">
    <id>governanceRegistryInstance</id>
    <dbConfig>governance_registry</dbConfig>
    <cacheId>regadmin@jdbc:mysql:10.20.30.41:3306/registrydb</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>regadmin@jdbc:mysql://10.20.30.42:3306/registrydb2</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:

```
<clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
enable="false"/>
```

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.

4. Copy 'MySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to \$G-REG_HOME/repository/components/lib in Carbon server instances of Foo product cluster.

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.

```

<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>
            <url>jdbc:mysql://10.20.30.42:3306/registrydb2</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 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,

```
<dbConfig name="governance_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
<dbConfig name="config_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG_CONFIG</dataSource>
</dbConfig>
```

Specify the remote Governance Registry instance with the following configuration:

```
<remoteInstance url="https://10.20.30.41:9443/registry">
    <id>governanceRegistryInstance</id>
    <dbConfig>governance_registry</dbConfig>
    <cacheId>regadmin@jdbc:mysql://10.20.30.41:3306/registrydb</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>regadmin@jdbc:mysql://10.20.30.42:3306/registrydb2</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:

```
<clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
enable="false"/>
```

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.

4. Copy 'MySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to \$G-REG_HOME/repository/components/lib in Carbon server instances of Bar product cluster.

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.

User Management

The 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.

This section provides the following information:

- [Introduction to User Management](#)
- [Adding and Managing Users and Roles](#)
- [Realm Configuration](#)
- [Changing the RDBMS](#)

- Configuring Primary User Stores
- 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.
- Implements management permission of the carbon console.

The user core is driven by the user-mgt.xml file found in: CARBON_HOME/repository/conf folder.

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 [Real m 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.

```
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 <DAS_HOME>/repository/conf/user-mgt.xml file.

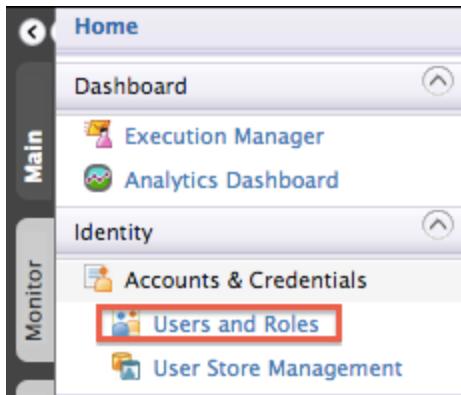
Go to the relevant topic listed below for details:

- [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 **Main** menu, click **Users and Roles**.



- Click **Users** from the **User Management** page that opens.

The **User** link is only visible to users with administrator permission.

- Click **Add New User**.
- 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](#).

Step 1 : Enter user name

Enter user name	
Domain	PRIMARY
User Name*	<input type="text"/>
Password*	<input type="password"/>
Password Repeat*	<input type="password"/>
<input type="button" value="Next >"/> <input type="button" value="Finish"/> <input type="button" value="Cancel"/>	

- If you want to add a user with the default Everyonerole, click **Finish** now. Else, click **Next** to define a user role other than the default .
- If you proceed to the next step, select the roles to be assigned to the user and **Finish**.
- The new user appears on the **Users** list.

Users

Enter user name pattern (* for all) *

Name	Actions
TestUser	
admin	

[Add New User](#)

- 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.

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 **Main** 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

The screenshot shows a web-based form titled "Enter role details". At the top, there is a dropdown menu labeled "Domain" with "PRIMARY" selected. Below it is a field labeled "Role Name*" which is currently empty. At the bottom of the form are three buttons: "Next >" (highlighted in grey), "Finish", and "Cancel".

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**.

For information on setting WSO2 DAS-specific permissions to user roles, see [WSO2 DAS-Specific User Permissions](#).

Step 2 : Select permissions to add to Role

All Permissions

- Admin Permissions
 - Configure
 - Login
 - Manage
 - Monitor
 - Logs
- Super Admin Permissions

< Back **Next >** Finish Cancel

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.

Name	Actions
admin	Users
Full Access	Rename Permissions Users Delete
everyone	Permissions

Add New Role

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 can not 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 **Main** menu, click **Users and Roles**.
2. The **User Management** page opens. Click on the **Change My Password**.

User Management

System User Store

Users

Roles

Change Password

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

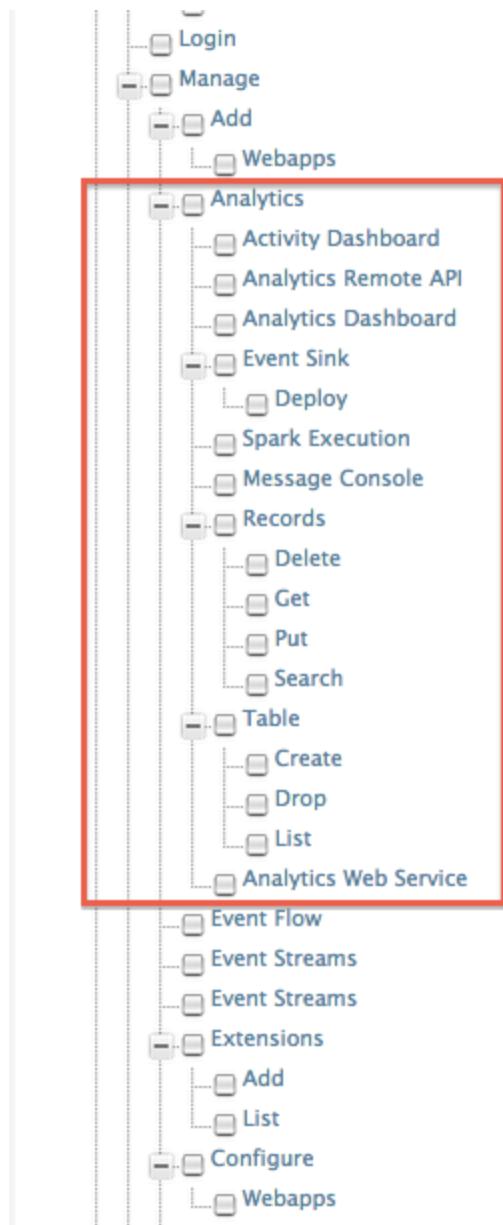
Follow the instructions below to delete a user.

Deleting a user cannot be undone.

1. On the **Main** tab in the management console, and then 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.

WSO2 DAS-Specific User Permissions

Set the WSO2 DAS-specific user permissions when [adding a new user role](#), or to an [existing user role](#), as shown below.



The above WSO2 DAS-specific permissions are explained below.

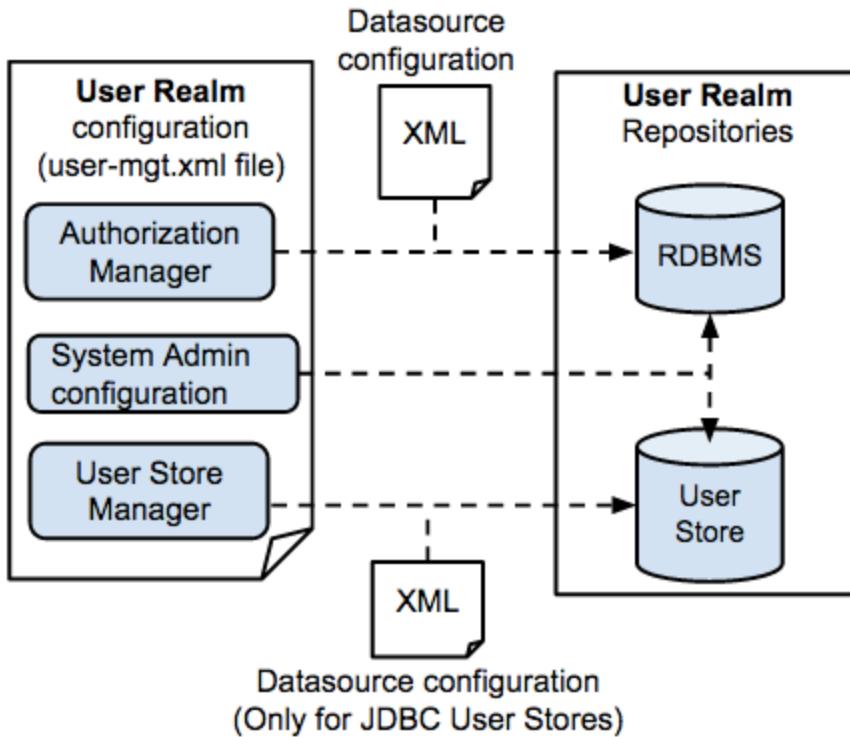
Permission	Description
Activity Dashboard	Required to access and use the activity explorer .
Analytics Remote API	Required to remotely connect to the DAS server, and access the APIs, and also to perform functions of the Data Explorer .

Analytics Dashboard	<p>Required to access and use the analytics dashboard.</p> <div style="border: 1px solid #ccc; padding: 5px;"> <p>To use the analytics dashboard, you also need the following permissions.</p> <ul style="list-style-type: none"> • GET (under Records) to retrieve records in the tables of the WSO2 DAS Data Access Layer (DAL) • List (under Table) to retrieve the tables of the WSO2 DAS Data Access Layer (DAL) • Analytics Web Service </div>
Event Sink Deploy	Required to deploy artifacts to event sink via CAR files or through event stream persistence .
Spark Execution	Required to execute Spark queries via the Spark console .
Message Console	Required to access the Data Explorer .
Records	Required to retrieve, insert, delete and search records in the tables of the WSO2 DAS Data Access Layer (DAL) , and also to perform functions of the Data Explorer . The Get permission is also required in order to use the analytics dashboard .
Table	Required to create, remove and retrieve tables of the WSO2 DAS Data Access Layer (DAL) , and also to perform functions of the Data Explorer . The List permission is also required in order to use the analytics dashboard .
Analytics Web Service	Required to remotely access the Analytics Web Service interface, and also to perform functions of the Data Explorer . This is also required in order to use the analytics dashboard .

Realm Configuration

The complete functionality and contents of the User Management module is called a **user realm**. The realm includes the user management classes, configurations and repositories that store information. Therefore, configuring the User Management functionality in a WSO2 product involves setting up the relevant repositories and updating the relevant configuration files.

The following diagram illustrates the required configurations and repositories:



See the following topics for more details:

- Configuring the system administrator
- Configuring the authorization manager

Configuring the system administrator

The **admin** user is the super tenant that will be able to manage all other users, roles and permissions in the system by using the management console of the product. Therefore, the user that should have admin permissions is required to be stored in the primary user store when you start the system for the first time. The documentation on setting up primary user stores will explain how to configure the administrator while configuring the user store. The information under this topic will explain the main configurations that are relevant to setting up the system administrator.

If the primary user store is read-only, you will be using a user ID and role that already exists in the user store, for the administrator. If the user store is read/write, you have the option of creating the administrator user in the user store as explained below. By default, the embedded H2 database (with read/write enabled) is used for both these purposes in WSO2 products.

Note the following key facts about the system administrator in your system:

- The admin user and role is always stored in the primary user store in your system.
- An administrator is configured for your system by default. This **admin** user is assigned to the **admin** role, which has all permissions enabled.
- The permissions assigned to the default **admin** role cannot be modified.

Updating the administrator

The `<Configuration>` section at the top of the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file allows you to configure the administrator user in your system as well as the RDBMS that will be used for storing information related to user authentication (i.e. role-based permissions).

```

<Realm>
    <Configuration>
        <AddAdmin>true</AddAdmin>
        <AdminRole>admin</AdminRole>
        <AdminUser>
            <UserName>admin</UserName>
            <Password>admin</Password>
        </AdminUser>
        <EveryOneRoleName>everyone</EveryOneRoleName> <!-- By default users in this role
see the registry root -->
        <Property name=""></Property>
        .....
    </Configuration>
    ...
</Realm>

```

Note the following regarding the configuration above.

Element	Description
<AddAdmin>	When <code>true</code> , this element creates the admin user based on the <code>AdminUser</code> element. It also indicates whether to create the specified admin user if it doesn't already exist. When connecting to an external read-only LDAP or Active Directory user store, this property needs to be <code>false</code> if an admin user and admin role exist within the user store. If the admin user and admin role do not exist in the user store, this value should be <code>true</code> , so that the role is added to the user management database. However, if the admin user is not there in the user store, we must add that user to the user store manually. If the <code>AddAdmin</code> value is set to <code>true</code> in this case, it will generate an exception.
<AdminRole>wso2admin</AdminRole>	This is the role that has all administrative privileges of the WSO2 product, so all users having this role are admins of the product. You can provide any meaningful name for this role. This role is created in the internal H2 database when the product starts. This role has permission to carry out any actions related to the Management Console. If the user store is read-only, this role is added to the system as a special internal role where users are from an external user store.
<AdminUser>	Configures the default administrator for the WSO2 product. If the user store is read-only, the admin user must exist in the user store or the system will not start. If the external user store is read-only, you must select a user already existing in the external user store and add it as the admin user that is defined in the <code><AdminUser></code> element. If the external user store is in read/write mode, and you set <code><AddAdmin></code> to <code>true</code> , the user you specify will be automatically created.
<UserName>	This is the username of the default administrator or super tenant of the user store. If the user store is read-only, the admin user MUST exist in the user store for the process to work.

<Password>	Do NOT put the password here but leave the default value. If the user store is read-only, this element and its value are ignored. This password is used only if the user store is read-write and the <code>AddAdmin</code> value is set to <code>true</code> .
<EveryOneRoleName>	Note that the password in the <code>user-mgt.xml</code> file is written to the primary user store when the server starts for the first time. Thereafter, the password will be validated from the primary user store and not from the <code>user-mgt.xml</code> file. Therefore, if you need to change the admin password stored in the user store, you cannot simply change the value in the <code>user-mgt.xml</code> file. To change the admin password, you must use the Change Password option from the management console.

Configuring the authorization manager

According to the default configuration in WSO2 products, the Users, Roles and Permissions are stored in the same repository (i.e., the default, embedded H2 database). However, you can change this configuration in such a way that the Users and Roles are stored in one repository (User Store) and the Permissions are stored in a separate repository. A user store can be a typical RDBMS, an LDAP or an external Active Directory.

The repository that stores Permissions should always be an RDBMS. The Authorization Manager configuration in the `user-mgt.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/` directory) connects the system to this RDBMS.

Follow the steps given below to set up and configure the Authorization Manager.

Step 1: Setting up the repository

By default, the embedded H2 database is used for storing permissions. You can change this as follows:

1. Change the default H2 database or set up another RDBMS for storing permissions.
2. When you set up an RDBMS for your system, it is necessary to create a corresponding datasource, which allows the system to connect to the database.
 - If you are replacing the default H2 database with a new RDBMS, update the `master-datasource.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/datasources/` directory) with the relevant information.
 - Alternatively, create a new XML file with the datasource information of your new RDBMS and store it in the same `<PRODUCT_HOME>/repository/conf/datasources/` directory.

Refer the [related topics](#) for detailed information on setting up databases and configuring datasources.

Step 2: Updating the user realm configurations

Once you have set up a new RDBMS and configured the datasource, the `user-mgt.xml` file (user realm configuration) should be updated as explained below.

1. Set up the database connection by update the datasource information using the `<Property>` element under `<Configuration>`. The jndi name of the datasource should be used to refer to the datasource. In the following example, the jndi name of the default datasource defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file is linked from the `user-mgt.xml` file.

```
<Realm>
    <Configuration>
        .....
        <Property name="dataSource">jdbc/WSO2CarbonDB</Property>
    </Configuration>
    ...
</Realm>
```

You can add more configurations using the `<Property>` element:

Property Name	Description
testOnBorrow	It is recommended to set this property to 'true' so that object connections will be validated before being borrowed from the JDBC pool. For this property to be effective, the validationQuery parameter in the <code><PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml</code> file should be a non-string value. This setting will avoid connection failures. See the section on performance tuning of WSO2 products for more information.

2. The default Authorization Manager section in the `user-mgt.xml` file is shown below. This can be updated accordingly.

```
<AuthorizationManager
    class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
    <Property name="AdminRoleManagementPermissions">/permission</Property>
    <Property name="AuthorizationCacheEnabled">true</Property>
</AuthorizationManager>
```

- The `org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager` class enables the Authorization Manager for your product.
- The `AdminRoleManagementPermissions` property sets the registry path where the authorization information (role-based permissions) are stored. Note that this links to the repository that you defined in Step 1.
- It is recommended to enable the `GetAllRolesOfUserEnabled` property in the `AuthorizationManager` as follows:

```
<Property name="GetAllRolesOfUserEnabled">true</Property>
```

Although 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 permission stats.

- By default, the rules linked to a permission (role name, action, resource) are not case sensitive. If you want to make them case sensitive, enable the following property:

```
<Property name="CaseSensitiveAuthorizationRules">true</Property>
```

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 <CARBON_HOME>/repository/components/lib.
2. Change values of properties given in on the [Realm Configuration](#) page appropriately.
3. Create the database by running the relevant script in <CARBON_HOME> /dbscript/ 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
```

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:

User store manager class	Description
org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager	Use ReadOnlyLDAPUsers external LDAP user stores.
org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Use ReadWriteLDAPUsers both read and write operations uncommented in the code in

org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use ActiveDirectoryUserStoreManager <pre>SELECT UM_USER_NAME FROM UM_USER_ATTRIBUTE.UM_ID WHERE UM_USER_ATTRIBUTE.UM_ID = ?</pre> <p>rectory Domain Service (AD LDS). This can be used to use AD as read-only you need to use the <code>ReadOnlyLDAPUserStore</code>.</p>
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBCUserStoreManager <p>stores.</p>

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.

```
<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.
- The class attribute for a read-only LDAP is `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager" />`

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 file.


```
<UserStoreManager
    class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
```
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.

```
<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>
      <Property name="SharedGroupNameAttribute">cn</Property>
      <Property
        name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property>
      <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>
    </UserStoreManager>
  </Realm>
</UserManager>
```

- a. Update the connection details to match your user store. For example:

```
<Property name="ConnectionURL">ldap://localhost:10389</Property>
```

- b. 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:

```
<Property name="ConnectionName">uid=AdminLDAP,ou=system</Property>
<Property name="ConnectionPassword">2010#Avrudu</Property>
```

- c. 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.

```
<Property name="UserSearchBase">ou=system</Property>
```

- d. 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:

```
<Property name="UserNameAttribute">uid</Property>
```

- e. 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:

```
<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.

- f. 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

```
<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>
```

- g. 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.

- 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:

```
<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
```

- Use the following class for Active Directory.

```
<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.

- 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.
- 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:

```

<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
    <Property
name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
    <Property
name="ConnectionURL">ldap://localhost:${Ports.EmbeddedLDAP.LDAPServerPort}</Property>
    <Property name="ConnectionName">uid=admin,ou=system</Property>
    <Property name="ConnectionPassword">admin</Property>
    <Property name="PasswordHashMethod">SHA</Property>
    <Property name="UserNameListFilter">(objectClass=person)</Property>
    <Property name="UserEntryObjectClass">wso2Person</Property>
    <Property name="UserSearchBase">ou=Users,dc=wso2,dc=org</Property>
    <Property
name="UserNameSearchFilter">(&amp;(objectClass=person)(uid=?))</Property>
    <Property name="UserNameAttribute">uid</Property>
    <Property name="PasswordJavaScriptRegEx">[\S]{5,30}</Property>
    <Property name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="UsernameJavaRegEx">^[^~!@#$%^*+={}\\|\\\\\\&lt;&gt;,\\\"]{3,30}$</Property>
    <Property name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="RolenameJavaRegEx">^[^~!@#$%^*+={}\\|\\\\\\&lt;&gt;,\\\"]{3,30}$</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="GroupEntryObjectClass">groupOfNames</Property>
    <Property
name="GroupNameSearchFilter">(&amp;(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>
    <Property name="UserDNPattern">uid={0},ou=Users,dc=wso2,dc=org</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.`

Active directory user store sample:

```

<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
    <Property
name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</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="ConnectionPassword">A1b2c3d4</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">^[\S]{3,30}\$</Property>
                    <Property name="PasswordJavaScriptRegEx">^[\S]{5,30}\$</Property>
            <Property name="RolenameJavaScriptRegEx">^[\S]{3,30}\$</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>
                        <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">(&& (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.JDBCAuthorizationManager">
<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

```
<Property name="UserNameAttribute">uid</Property>
```

```
<Property name="UserNameAttribute">sAMAccountName</Property>
```

4. The following code snippet represents reading roles based on a backlink attribute. This is used by the Active Directory.

```
<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 user 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:

```
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
```

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:

```

<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="ReadGroups">true</Property>
    <Property name="WriteGroups">true</Property>
    <Property name="UsernameJavaRegEx">^\{3,30\}$</Property>
    <Property name="UsernameJavaScriptRegEx">^\{3,30\}$</Property>
    <Property name="UsernameJavaRegExViolationErrorMsg">Username pattern policy violated</Property>
    <Property name="PasswordJavaRegEx">^\{5,30\}$</Property>
    <Property name="PasswordJavaScriptRegEx">^\{5,30\}$</Property>
    <Property name="PasswordJavaRegExViolationErrorMsg">Password length should be within 5 to 30 characters</Property>
    <Property name="RolenameJavaRegEx">^\{3,30\}$</Property>
    <Property name="RolenameJavaScriptRegEx">^\{3,30\}$</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:

```

<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="userNmae">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="PasswordDigest">SHA-256</Property>
    <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="UserRoleSQL">SELECT UM_ROLE_NAME FROM UM_USER_ROLE, UM_ROLE, UM_USER WHERE UM_USER.UM_USER_NAME=? AND UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID AND UM_USER_ROLE.UM_TENANT_ID=? AND UM_ROLE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
<Property name="UserSharedRolesSQL">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 INNER JOIN UM_ROLE ON UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.UM_ID WHERE UM_USER.UM_USER_NAME = ? AND UM_SHARED_USER_ROLE.UM_ROLE_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=" GetUserListOfRoleSQL">SELECT UM_USER_NAME FROM UM_USER_ROLE, UM_ROLE, UM_USER WHERE UM_ROLE.UM_ROLE_NAME=? AND UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID AND UM_USER_ROLE.UM_TENANT_ID=? AND UM_ROLE.UM_TENANT_ID=? AND UM_USER.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 INNER JOIN UM_ROLE ON UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.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</Property>
<Property name=" IsUserExistingSQL">SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
<Property name=" GetUserPropertiesForProfileSQL">SELECT UM_ATTR_NAME, UM_ATTR_VALUE FROM UM_USER_ATTRIBUTE, UM_USER WHERE UM_USER.UM_ID = UM_USER_ATTRIBUTE.UM_USER_ID AND UM_USER_ATTRIBUTE.UM_USER_ID AND UM_USER.UM_USER_NAME=? AND UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
<Property name=" GetUserPropertyForProfileSQL">SELECT UM_ATTR_VALUE FROM UM_USER_ATTRIBUTE, UM_USER WHERE UM_USER.UM_ID = UM_USER_ATTRIBUTE.UM_USER_ID AND UM_USER.UM_USER_NAME=? AND UM_ATTR_NAME=? AND UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
<Property name=" GetUserLisForPropertySQL">SELECT UM_USER_NAME FROM UM_USER, UM_USER_ATTRIBUTE WHERE UM_USER_ATTRIBUTE.UM_USER_ID = UM_USER.UM_ID AND UM_USER_ATTRIBUTE.UM_ATTR_NAME=? AND UM_USER_ATTRIBUTE.UM_ATTR_VALUE=? AND UM_USER_ATTRIBUTE.UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?SELECT UM_USER_NAME FROM UM_USER, UM_USER_ATTRIBUTE WHERE UM_USER_ATTRIBUTE.UM_USER_ID = UM_USER.UM_ID AND UM_USER_ATTRIBUTE.UM_ATTR_NAME
```

```

=? AND UM_USER_ATTRIBUTE.UM_ATTR_VALUE LIKE ? AND
UM_USER_ATTRIBUTE.UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND
UM_USER.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="AddUserToRoleSQL">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, UM_ROLE_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="RemoveUserFromSharedRoleSQL">DELETE FROM
UM_SHARED_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_USER_TENANT_ID=? AND
UM_ROLE_TENANT_ID = ?</Property>
<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="OnDeleteUserRemoveUserRoleMappingSQL">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_TENANT_ID=?</Property>
<Property name="OnDeleteUserRemoveUserAttributeSQL">DELETE 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="UpdateUserPasswordSQL">UPDATE UM_USER SET UM_USER_PASSWORD=?
, UM_SALT_VALUE=?, UM_REQUIRE_CHANGE=?, UM_CHANGED_TIME=? 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="UpdateUserPropertySQL">UPDATE UM_USER_ATTRIBUTE SET
UM_ATTR_VALUE=? WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=?
AND UM_TENANT_ID=?) AND UM_ATTR_NAME=? AND UM_PROFILE_ID=? AND
UM_TENANT_ID=?</Property>
<Property name="DeleteUserPropertySQL">DELETE FROM UM_USER_ATTRIBUTE WHERE
UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?)
AND UM_ATTR_NAME=? AND UM_PROFILE_ID=? 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="AddUserToRoleSQL-mssql">INSERT INTO UM_USER_ROLE
(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>
<Property name="AddUserToRoleSQL-openedge">INSERT INTO UM_USER_ROLE
(UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) SELECT UU.UM_ID, UR.UM_ID, ? FROM UM_USER
UU, UM_ROLE UR WHERE UU.UM_USER_NAME=? AND UU.UM_TENANT_ID=? AND
UR.UM_ROLE_NAME=? AND UR.UM_TENANT_ID=?</Property>
<Property name="AddRoleToUserSQL-openedge">INSERT INTO UM_USER_ROLE
(UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) SELECT UR.UM_ID, UU.UM_ID, ? FROM UM_ROLE
UR, UM_USER UU WHERE UR.UM_ROLE_NAME=? AND UR.UM_TENANT_ID=? AND
UU.UM_USER_NAME=? AND UU.UM_TENANT_ID=?</Property>
<Property name="AddUserPropertySQL-openedge">INSERT INTO UM_USER_ATTRIBUTE
(UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) SELECT
UM_ID, ?, ?, ?, ? FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>

```

```
<Property name="DomainName">wso2.org</Property>
<Property name="Description"/>
</UserStoreManager>
```

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>.

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:

```
<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.

Property name	Description
MaxUserNameListLength	Controls the number of users listed in the user store of a WSO2 product. If you want to list them all, Setting this property to 0 displays all users.
ConnectionURL	Connection URL to the user store server. In the case of default connection, reference to that port is included in this configuration.
ConnectionName	The username used to connect to the database and perform various operations in the user store or have an administrator role in the WSO2 product that you can use to manage users' attributes and to perform search operations on the user store. This attribute is mandatory.
ConnectionPassword	Password for the ConnectionName user.
DisplayNameAttribute	This is an optional property. The Display Name Attribute is the name of the user displayed in the user management console (Go to Configuration -> Users tab).
PasswordHashMethod	Password hash method to use when storing user entries in the user store.
UserNameListFilter	Filtering criteria for listing all the user entries in the user store. This filter is applied to the search operation. In this case, the search operation only provides the objects created from this filter.
UserEntryObjectClass	Object class used to construct user entries. By default, it is a custom class.
UserSearchBase	DN of the context or object under which the user entries are stored. When performing user store searches for users, it will start from this location of the directory.
	Different databases have different search bases.
UserNameSearchFilter	Filtering criteria used to search for a particular user entry.
UserNameAttribute	The attribute used for uniquely identifying a user entry. Users can be identified by this attribute.
	The name of the attribute is considered as the username.

UsernameWithEmailJavaScriptRegEx	This property defines the JavaScript regular expression pattern which is used to validate user names in the configuration file. If you need to support both email as a user name and a plain user name, you can define two separate regular expressions.
PasswordJavaScriptRegEx	Policy that defines the password format.
UsernameJavaScriptRegEx	The regular expression used by the front-end components for user name validation.
UsernameJavaRegEx	A regular expression to validate usernames. By default, strings have to contain at least one character and provide ranges of alphabets, numbers and also ranges of ASCII values.
RolenameJavaScriptRegEx	The regular expression used by the front-end components for role name validation.
RolenameJavaRegEx	A regular expression used to validate role names. By default, string has to contain at least one character.
ReadGroups	Specifies whether groups should be read from the user store. If this is set to true, then the following group configurations are NOT mandatory.
WriteGroups	Specifies whether groups should be written to user store.
EmptyRolesAllowed	Specifies whether the underlying user store allows empty groups to exist or not. By default, it is false, which means that empty groups are allowed to be created. Usually LDAP servers do not allow empty groups.
GroupSearchBase	DN of the context under which user entries are stored in the user store.
GroupSearchFilter	The query used to search for groups.
GroupNameListFilter	Filtering criteria for listing all the group entries in the user store. Group search operation only returns objects created from this class.
GroupEntryObjectClass	Object class used to construct group entries.
GroupNameSearchFilter	Filtering criteria used to search for a particular group entry.
GroupNameAttribute	Attribute used for uniquely identifying a user entry. This attribute is mandatory.
MembershipAttribute	Attribute used to define members of groups.
UserRolesCacheEnabled	This is to indicate whether to cache the role list of a user. By default, it is true. If it is false, then changes made to the roles through external means and those changes should be instantly reflected in the user store.
UserDNPattern	(LDAP) The pattern for the user's DN, which can be defined to improve performance. Defining a UserDNPattern provides more impact on performance than defining a UserObjectClass.
ReplaceEscapeCharactersAtUserLogin	(LDAP) If the user name has special characters it replaces it to valid characters.

TenantManager	Includes the location of the tenant manager.
ReadOnly	(LDAP and JDBC) Indicates whether the user store of this realm is read-only.
IsEmailUserName	(JDBC) Indicates whether the user's email is used as their username.
DomainCalculation	(JDBC) Can be either default or custom (this applies when the realm is configured with JDBC).
PasswordDigest	(JDBC) Digesting algorithm of the password. Has values such as, F1, SHA-1, SHA-256, etc.
StoreSaltedPassword	(JDBC) Indicates whether to salt the password.
UserNameUniqueAcrossTenants	(JDBC) An attribute used for multi-tenancy.
PasswordJavaRegEx	(LDAP and JDBC) A regular expression to validate passwords. Only alphanumeric characters are allowed.
PasswordJavaScriptRegEx	The regular expression used by the front-end components for password validation.
UsernameJavaRegEx	A regular expression to validate usernames. By default, strings have to be alphanumeric.
UsernameJavaScriptRegEx	The regular expression used by the front-end components for user name validation.
RolenameJavaRegEx	A regular expression to validate role names. By default, strings have to be alphanumeric.
RolenameJavaScriptRegEx	The regular expression used by the front-end components for role name validation.
MultiTenantRealmConfigBuilder	Tenant Manager specific realm config parameter. Can be used to build a configuration for a specific tenant.
SharedGroupEnabled	This property is used to enable/disable the shared role functionality.
SharedGroupSearchBase	Shared roles are created for other tenants to access under the mentioned search base.
SharedTenantObjectClass	Object class for the shared groups created.
SharedTenantNameAttribute	Name attribute for the shared group.
SharedTenantNameListFilter	This is currently not used.

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](#)

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

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:

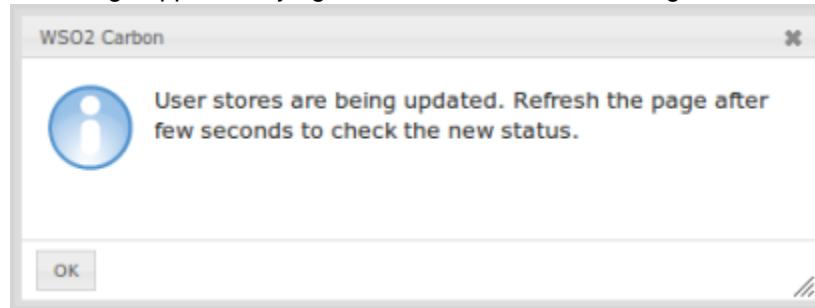
User store manager	Description
org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager	Use ReadOnlyLDAPUserStoreManager class to use Read Only LDAP user stores.
org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Use ReadWriteLDAPUserStoreManager class to use Read Write LDAP user stores.
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use ActiveDirectoryUserStoreManager class to use Active Directory Domain Service (AD DS) or Lightweight Directory Services (LDS). This can be used as a read-only, you must use the <code>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</code> class.
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBCUserStoreManager class to use JDBC based user stores. This can be configured for multiple databases. The configuration property: <Property name="UserManager" value="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager" />

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.

Property Name	Property Value	Description
ConnectionName*	uid=admin,ou=system	This should be the DN (Distinguish Name) of the admin user in LDAP
ConnectionURL*	localhost:\${Ports.EmbeddedLDAP.LDAPServerPort}	Connection URL for the user store
ConnectionPassword*	*****	Password of the admin user
UserSearchBase*	ou=Users,dc=wso2,dc=org	DN of the context under which user entries are stored in LDAP
Disabled*	<input type="checkbox"/>	Whether user store is disabled
UserNameListFilter*	(objectClass=person)	Filtering criteria for listing all the user entries in LDAP
UserNameAttribute*	uid	Attribute used for uniquely identifying a user entry. Users can be authenticated using their email address, uid and etc.
UserNameSearchFilter*	(&(objectClass=person)(uid=?))	Filtering criteria for searching a particular user entry
UserEntryObjectClass*	wso2Person	Object Class used to construct user entries

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.



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.

9. Refresh the page after a few seconds to check the status.
10. If the new user store is successfully added, it will appear in the **User Store Management** page.
11. 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 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 `wso`

- `2_com.xml`.
- One file only contains the definition for one user store domain.

Related topics

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 Data Analytics Server 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 explained 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 (`wso2s`

erver.bat) to the product's distribution home (e.g., -Dcarbon.home=C:\Users\VM\Desktop\wso2da-s-3.0.0). 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,

You can add a new local repository or a remote repository

Name: *	<input type="text" value="My_Repo"/>
Location:	<input checked="" type="radio"/> URL <input type="text" value="http://dist.wso2.org/p2/carbon/releases/3.2.0"/> <input type="radio"/> Local <input type="text"/>
<small>e.g. http://dist.wso2.org/p2/carbon/releases/3.0.0</small> <small>e.g. C:/user/repo, /home/user/p2-repo</small>	
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

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.

Add/edit/remove/disable repositories which contains Features.

Available Repositories:				
Name	Location	Enabled	Actions	
p1	http://dist.wso2.org/p2/carbon/releases/3.0.0	Enabled	Edit	Remove
my_repository	http://dist.wso2.org/p2/carbon/releases/3.2.0	Enabled	Edit	Remove

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.

The screenshot shows the 'Feature Management' interface with the 'Available Features' tab selected. A red box highlights the 'Available Features' tab. The 'Repository' dropdown is set to 'Test - http://dist.wso2.org/p2/carbon/releases/3.2.0'. Below it are filter options: 'Show only the latest versions' (unchecked) and 'Group features by category' (checked). A 'Find Features' button is at the bottom.

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.

The screenshot shows the 'Find Features' search results. At the top, there are filter options: 'Repository' set to 'bambooRepo - http://wso2.org/bamboo/artifact/WS02CARBON-P2REPO/JOB1/build-14', 'Filter by feature name:' (empty), 'Show only the latest versions' (unchecked), and 'Group features by category' (checked, highlighted with a red box). Below these are 'Find Features' and 'Install' buttons. The main table lists features under the 'Features' tab. A vertical red line highlights the first few rows of the table.

Features	Version	Actions
<input type="checkbox"/> Application Server		
<input checked="" type="checkbox"/> Data Services Server		
<input checked="" type="checkbox"/> XKMS Management	4.0.0.SNAPSHOT	More Info
<input checked="" type="checkbox"/> Xfer Module	4.0.0.SNAPSHOT	More Info
<input checked="" type="checkbox"/> WSDL Tools	4.0.0.SNAPSHOT	More Info
<input checked="" type="checkbox"/> WS-Discovery Core	4.0.0.SNAPSHOT	More Info
<input checked="" type="checkbox"/> Tryit	4.0.0.SNAPSHOT	More Info
<input checked="" type="checkbox"/> Transport Management	4.0.0.SNAPSHOT	More Info
<input checked="" type="checkbox"/> Transaction-Manager Server	4.0.0.SNAPSHOT	More Info

- 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.

- From the list of features that appear, select the ones you want to add and click **Install**.
- The **Install Details** page appears. Verify the provided information and click **Next**.
- Read and accept the terms of license agreement.
- The installation process starts. It may take a few minutes to download the necessary components.
- Once the installation process is complete, click **Finish** and restart the server for the changes to take effect.
- Go to the **Installed Features** tab to browse through the list of installed features.

Turning your product to a back-end/front-end server

- 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:

The screenshot shows the 'Uninstall' list for front-end features. At the top, there is a dropdown menu set to 'Front-end' (highlighted with a red box) and a search bar. Below these are 'Select all in this page' and 'Uninstall' buttons. The main table lists features under the 'Features' tab. A vertical red line highlights the first two rows of the table.

Features	Version	Actions
<input checked="" type="checkbox"/> Admin Console UI	3.2.0	More Info
<input checked="" type="checkbox"/> Axis2 Service Hosting UI	3.2.0	More Info

This lists all the front-end features that are currently installed in the system.

- 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,

The screenshot shows the 'Feature Management' interface with the 'Installation History' tab selected. At the top, there are tabs for 'Available Features', 'Installed Features', 'Installation History' (which is active), and 'Repository Management'. Below the tabs, a heading 'Installation History' is followed by a sub-instruction: 'This page lists the history of provisioning operations performed on the system. Click on a configuration to view more details'. A table lists several configurations with their respective dates and times:

Configuration	Date
Previous Configurations	
Current Configuration	
August 26, 2013 at 13:42:54 IST	
August 25, 2013 at 13:28:40 IST	
August 25, 2013 at 13:28:34 IST	
August 25, 2013 at 13:28:24 IST	

4. Click on a configuration to view its details. For example,

The screenshot shows the 'Feature Management' interface with the 'Reverting to Previous Configuration' dialog open. The title bar says 'Reverting to Previous Configuration - August 26, 2013 at 13:42:54 IST'. A sub-instruction below the title reads: 'Reverting to the Selected Configuration, would result in performing following actions on the Current Configuration.' It states that the following features will be uninstalled:

Name	Version	ID	Provider
WSO2 Carbon – Doc Request Processor Feature	4.2.0	org.wso2.carbon.docrequestprocessor.feature.group	WSO2 Inc.
WSO2 Carbon – API Store Feature	4.2.0	org.wso2.carbon.apimgt.store.feature.group	WSO2 Inc.

At the bottom of the dialog are two buttons: '< Back' and 'Revert'.

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 `-D osgiConsole 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>
For example:
osgi> getInstallationHistory 1376933879416
```

The output gives you details similar to the following:

```
-- 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.
```

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`
- 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:

The screenshot shows the 'Logging Configuration' page with three main sections:

- Global Log4J Configuration:** Contains fields for 'Log Level' (set to 'INFO') and 'Log Pattern' (set to '[%d] %5p - %x %m [%c]%n'). Below these are 'Update' and 'Restore Defaults' buttons.
- Configure Log4J Appenders:** Contains fields for 'Name' (set to 'CARBON_CONSOLE'), 'Log Pattern' (set to '[%d] %P%5p [%c] - %x %m%n'), and 'Threshold' (set to 'DEBUG'). Below these are 'Update' and 'Restore Defaults' buttons.
- Configure Log4J Loggers:** Contains a 'Filter Loggers by' input field with dropdown options 'Starts With' and 'Contains'.

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.

Restore Defaults button allows to overwrite the Registry with the logging configurations specified in `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
- org.wso2.carbon.dataservices.task.internal.DSTaskServiceComponent

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,

Logger	Parent Logger	Level	Additivity
org.wso2.carbon.dataservices.core.DBDeployer	org.wso2	DEBUG	True
org.wso2.carbon.dataservices.core.DBInOnlyMessageReceiver	org.wso2	INFO	True
org.wso2.carbon.dataservices.core.DBInOutMessageReceiver	org.wso2	INFO	True
org.wso2.carbon.dataservices.core.DBUtils	org.wso2	INFO	True
org.wso2.carbon.dataservices.core.admin.DataServiceAdmin	org.wso2	INFO	True
org.wso2.carbon.dataservices.core.admin.DataServiceFileUploader	org.wso2	INFO	True

Datasources

A datasource is a connection set up to a storage of data, such as a database or a data file, from a server. A datasource provides information that a server can use to connect to a storage of data.

- [Enabling datasource management](#)
- [Adding datasources](#)
- [WSO2 DAS default datasources](#)
- [Datasource types](#)

Enabling datasource management

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

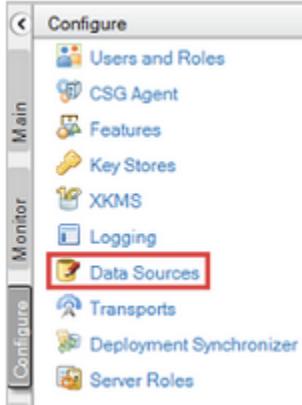
If datasource management capability is not included in your product by default, add it by installing the above feature. For instructions on the datasource management feature, see [Feature Management](#).

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. Select 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
4. 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.

You can view, edit, and delete the datasources in your product instance by clicking **Data Sources** in the **Configure** menu of the product management console. However, you cannot edit or delete the default <WSO2_CARBON_DB> datasource.

WSO2 DAS default datasources

The default **RDBMS** type datasources, which are shipped with WSO2 DAS are defined in the <DAS_HOME>/repository/conf/datasources/ directory as follows.

Home > Configure > Datasources Help

Datasources

Available Datasources		
Datasource	Status	Action
WSO2_ANALYTICS_EVENT_STORE_DB	ACTIVE	View
WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB	ACTIVE	View
WSO2_METRICS_DB	ACTIVE	View
WSO2_CARBON_DB	ACTIVE	View
WSO2ML_DB	ACTIVE	View

[Add Datasource](#)

Master datasources

The `WSO2_CARBON_DB` master datasource, which is defined in the <DAS_HOME>/repository/conf/master-d

`tasources.xml` file is used to store data of registry and user manager of WSO2 DAS in a RDBMS.

By default, this points to the in-built H2 database of the DAS. Replace the `url`, `username`, `password` and `driverClassName` settings with your custom values and also the other values accordingly in the master `-datasources.xml` file as shown in the below example, to change the underlying RDBMS to which this datasource connects. For information on changing the underlying database type, see [Working with Databases](#).

```
<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=600
00</url>
        <username>wso2carbon</username>
        <password>wso2carbon</password>
        <driverClassName>org.h2.Driver</driverClassName>
        <maxActive>50</maxActive>
        <maxWait>60000</maxWait>
        <testOnBorrow>true</testOnBorrow>
        <validationQuery>SELECT 1</validationQuery>
        <validationInterval>30000</validationInterval>
        <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
    </definition>
</datasource>
```

The datasource configuration options are as follows. For more information on these configuration properties, see [Apache TomEE Documentation](#).

Parameter	Description
<code><url></code>	The URL of the database.
<code><username></code>	The name of the database user.
<code><password></code>	The password of the database user.
<code><driverClassName></code>	The class name of the database driver.
<code><maxActive></code>	The maximum number of active connections that can be allocated from this pool at the same time, or enter a negative value for no limit.
<code><maxWait></code>	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.

<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>	The default auto-commit state of new connections.

Analytics datasources

Following analytics datasources are defined in the <DAS_HOME>/repository/conf/datasources/analytics-datasources.xml file. For more information on the usage and configuration of these datasources, see [Configuring Data Persistence](#).

Datasource Name	Description
WSO2_ANALYTICS_EVENT_STORE_DB	This datasource is used by default to connect to the data store of the Analytics Record Store which stores event definitions.
WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB	This datasource is used by default to connect to the data store of the Analytics Record Store which stores processed data.

Other datasources

The following table describes the other datasources that are shipped with WSO2 DAS by default.

Datasource Name	Description
WSO2_METRICS_DB	This datasource is used to store metrics related data of WSO2 DAS.
WSO2_ML_DB	This datasource is used to store data relating to the Machine Learner component.

Datasource types

The following sections explain the datasource types that are used in DAS.

- [Configuring an RDBMS Datasource](#)
- [Configuring a Cassandra Datasource](#)
- [Configuring a HBase Datasource](#)
- [Configuring a HDFS Datasource](#)
- [Configuring a Custom Datasource](#)

Configuring an RDBMS Datasource

When adding a datasource, if you select RDBMS as the datasource type, the following screen appears:

New Data Source

New Data Source

Data Source Type*	RDBMS
Name*	
Description	
Data Source Provider*	default
Driver*	
URL*	
User Name	
Password	
<input type="checkbox"/> Expose as a JNDI Data Source <input type="checkbox"/> Data Source Configuration Parameters	
<input type="button" value="Test Connection"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/>	

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-5.5.2-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 Souce:** 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.

When adding an RDBMS datasource, be sure to copy the JDBC driver JAR file for your database to <PRODUCT_HOME>/repository/components/lib/ directory.

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:

The screenshot shows the 'New Data Source' configuration dialog. The 'Data Source Type' is set to 'RDBMS'. The 'Name' field contains 'rdbmsdatasource'. The 'Description' field is 'RDBMS Data Source'. The 'Data Source Provider' dropdown is set to 'default' (highlighted with a red box). The 'Driver' field is 'com.mysql.jdbc.Driver'. The 'URL' field is 'jdbc:mysql://localhost:3306/test'. The 'User Name' field is 'root'. The 'Password' field is masked with four dots. Below the main form, there are two expandable sections: 'Expose as a JNDI Data Source' and 'Data Source Configuration Parameters'. At the bottom are 'Test Connection', 'Save', and 'Cancel' buttons.

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:

The screenshot shows the 'New Data Source' configuration dialog for an external provider. The 'Data Source Type' is 'RDBMS'. The 'Name' field is 'rdbmsdatasource'. The 'Description' field is 'RDBMS Data Source'. The 'Data Source Provider' dropdown is set to 'External Data Source' (highlighted with a red box). The 'Data Source Class Name' field is 'lbc.jdbc2.optional.MysqlXADataSource'. Below this, there is a table titled 'Add Property' with three rows: 'url' (value: ':mysql://localhost:3306/test'), 'user' (value: 'root'), and 'password' (value: 'root'). The table has columns for 'Name', 'Value', and 'Action' (with delete icons). Below the main form, there are two expandable sections: 'Expose as a JNDI Data Source' and 'Data Source Configuration Parameters'. At the bottom are 'Test Connection', 'Save', and 'Cancel' buttons.

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:

New Data Source

Data Source Type* RDBMS

Name*

Description

Data Source Provider* default

Driver*

URL*

User Name admin

Password *****

Expose as a JNDI Data Source

Name

Use Data Source Factory

JNDI Properties

Data Source Configuration Parameters

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

New Data Source

DataSource Id*	oracle-ds
Data Source Type*	RDBMS
Database Engine*	Oracle
Driver Class*	oracle.jdbc.driver.OracleDriver
URL*	jdbc:oracle:[drivertype]:[username/password]@[host]:[port]/[database]
User Name	
Password	
<input checked="" type="checkbox"/> Data source configuration parameters	
Transaction Isolation	TRANSACTION_UNKNOWN
Initial Size	
Max. Active	
Max. Idle	
Min. Idle	
Max. Wait	
Validation Query	
Test On Return	false
Test On Borrow	true
Test While Idle	false
Time Between Eviction Runs Mills	
Minimum Evictable Idle Time	
Remove Abandoned	false
Remove Abandoned Timeout	
Log Abandoned	false
Default Auto Commit	false
Default Read Only	false
Default Catalog	
Validator Class Name	
Connection Properties	
Init SQL	
JDBC Interceptors	
Validation Interval	
JMX Enabled	false
Fair Queue	false
Abandon When Percentage Full	
Max Age	
Use Equals	false

The screenshot shows a configuration dialog for a JDBC connection pool. At the top, there are two dropdown menus: 'Suspect Timeout' set to '10000' and 'Alternate User Name Allowed' set to 'false'. Below these are three buttons: 'Test Connection', 'Save', and 'Cancel'. The main area contains a table with various configuration parameters and their descriptions.

Following are descriptions of the parameters you can configure. For more details on datasource configuration parameters, see [ApacheTomcat JDBC Connection Pool guide](#).

Parameter name	Description
Transaction isolation	<p>The default TransactionIsolation state of connections created by this pool are as follows:</p> <ul style="list-style-type: none"> • TRANSACTION_UNKNOWN • TRANSACTION_NONE • TRANSACTION_READ_COMMITTED • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_REPEATABLE_READ • TRANSACTION_SERIALIZABLE
Initial Size (int)	The initial number of connections created, when the pool is started. Default value is zero.
Max. Active (int)	Maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.
Max. Idle (int)	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>)
Min. Idle (int)	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> .
Max. Wait (int)	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).
Validation Query (String)	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).
Test On Return (boolean)	<p>Used to indicate if objects will be validated before returned to the pool. The default value is false.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.</p> </div>
Test On Borrow (boolean)	<p>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.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>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> .</p> </div>

Test While Idle (boolean)	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 timeBetweenEvictionRunsMillis .
	For a true value to have any effect, the validationQuery parameter must be set to a non-null string.
Time Between Eviction Runs Mills (int)	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).
Minimum Evictable Idle Time (int)	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).
Remove Abandoned (boolean)	Flag to remove abandoned connections if they exceed the removeAbandonedTimeout . If set to true, a connection is considered abandoned and eligible for removal, if it has been in use longer than the removeAbandonedTimeout . Setting this to true can recover database connections from applications that fail to close a connection. For more information, see logAbandoned . The default value is false.
Remove Abandoned Timeout (int)	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.
Log Abandoned (boolean)	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.
Auto Commit (boolean)	The default auto-commit state of connections created by this pool. If not set, default is JDBC driver default. If not set, then the setAutoCommit method will not be called.
Default Read Only (boolean)	The default read-only state of connections created by this pool. If not set then the setReadOnly method will not be called. (Some drivers don't support read only mode. For example: Informix)
Default Catalog (String)	The default catalog of connections created by this pool.
Validator Class Name (String)	The name of a class which implements the <code>org.apache.tomcat.jdbc.pool.Validator</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> .

Connection Properties (String)	<p>Connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be [propertyName=property;]*. The default value is null.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>The <code>user</code> and <code>password</code> properties will be passed explicitly, so that they do not need to be included here.</p> </div>
Init SQL	Ability to run a SQL statement exactly once, when the connection is created.
JDBC Interceptors	Flexible and pluggable interceptors to create any customizations around the pool, the query execution and the result set handling.
Validation Interval (long)	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).
JMX Enabled (boolean)	Register the pool with JMX or not. The default value is true.
Fair Queue (boolean)	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 (property <code>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.
Abandon When Percentage Full (int)	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.
Max Age (long)	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.
Use Equals (boolean)	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.
Suspect Timeout (int)	Timeout value in seconds. Default value is zero. Similar to the <code>removeAbandonedTimeout</code> value, but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if <code>logAbandoned</code> 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.

<p>Alternate User Name Allowed (boolean)</p>	<p>By default, the <code>jdbc-pool</code> will ignore the <code>DataSource.getConnection(username, password)</code> call, and simply return a previously pooled connection under the globally configured properties <code>username</code> and <code>password</code>, for performance reasons.</p> <p>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 <code>DataSource.getConnection(username, password)</code> call, simply set the property <code>alternateUsernameAllowed</code>, 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.</p>
-----------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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.

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

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 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>
```

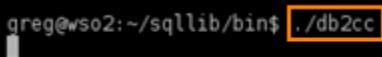
For example:

```
db2 => connect to regdb user greg using 18091980
Database Connection Information
Database server      = DB2/LINUX 9.7.4
SQL authorization ID = GREG
Local database alias = REGDB
db2 => GRANT DBADM, CREATETAB, BINDADD, CONNECT, CREATE_NOT_FENCED, IMPLICIT_SCHEMA, LOAD ON DATABASE TO USER user
DB20000I  The SQL command completed successfully.
db2 => █
```

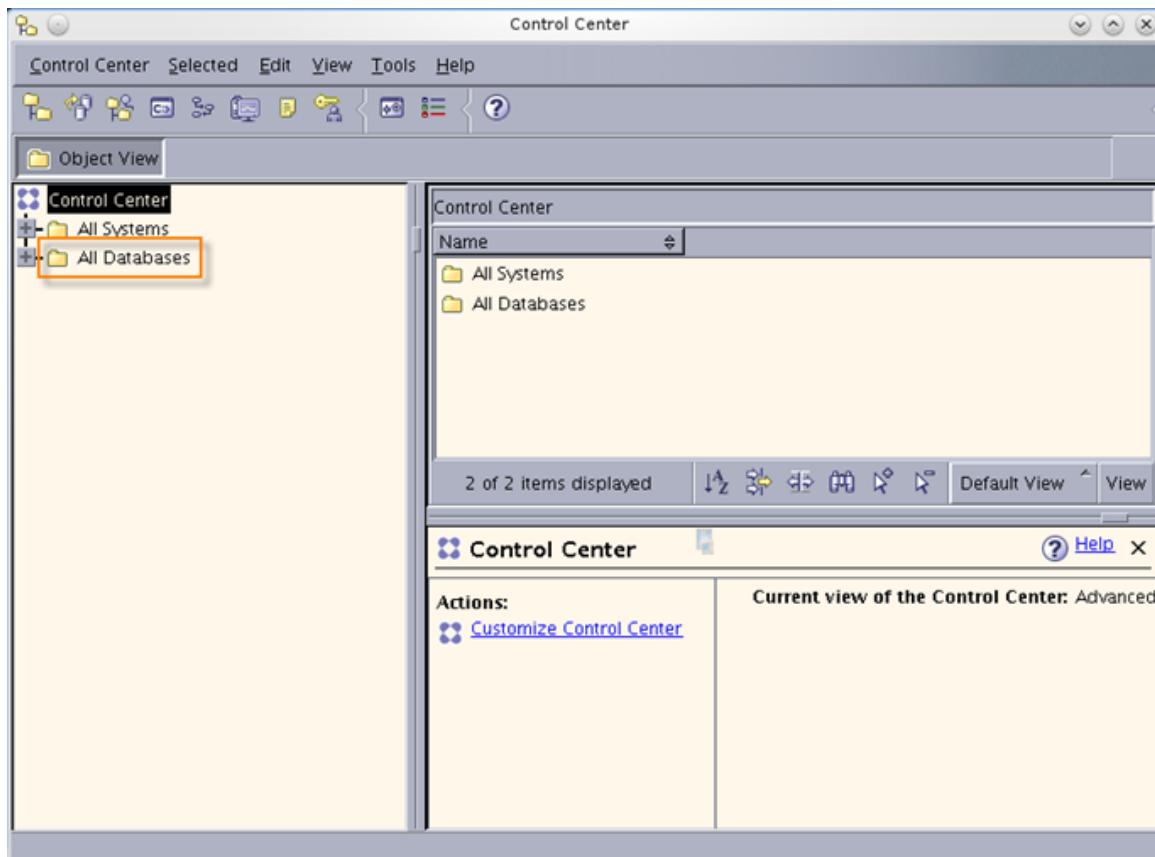
For more information on DB2 commands, see the [DB2 Express-C Guide](#).

Using the DB2 control center

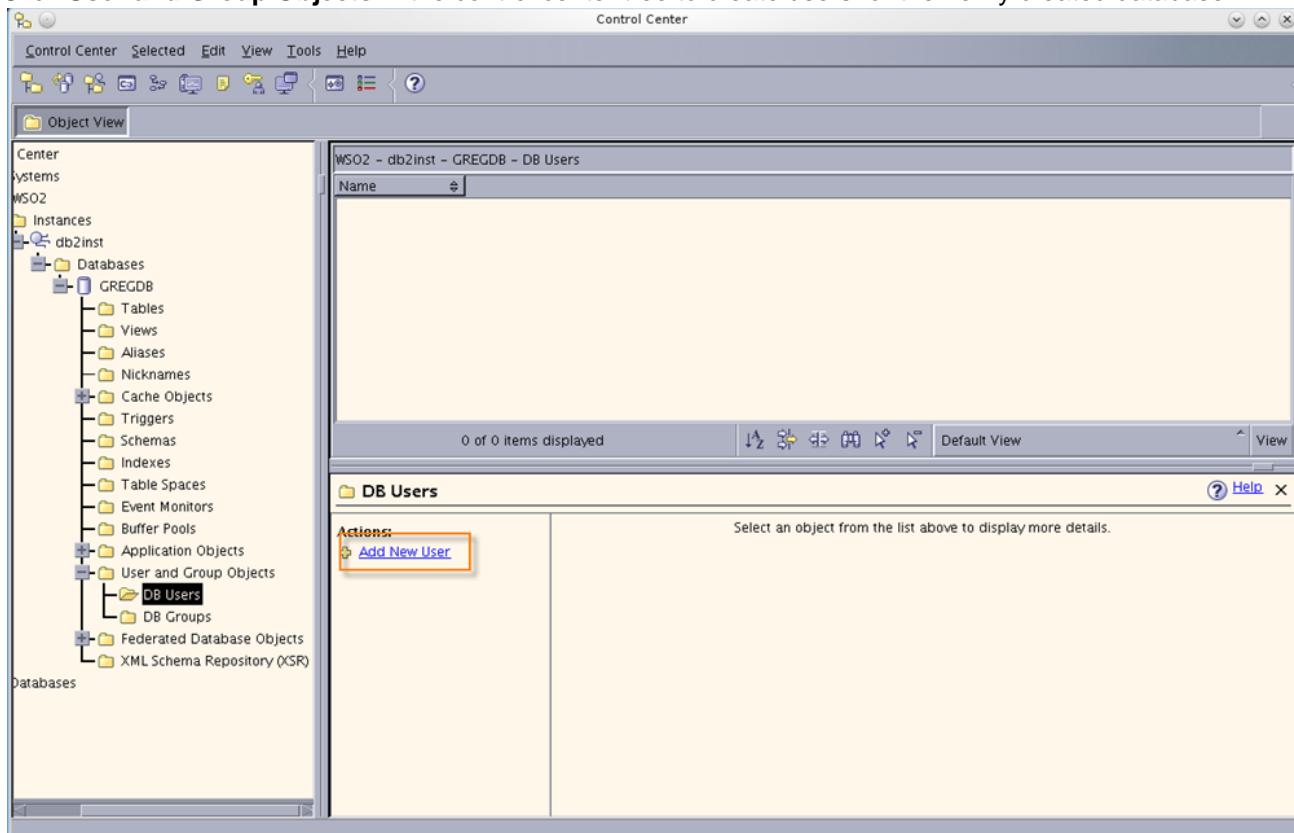
1. Open the DB2 control center using the db2cc command as follows:



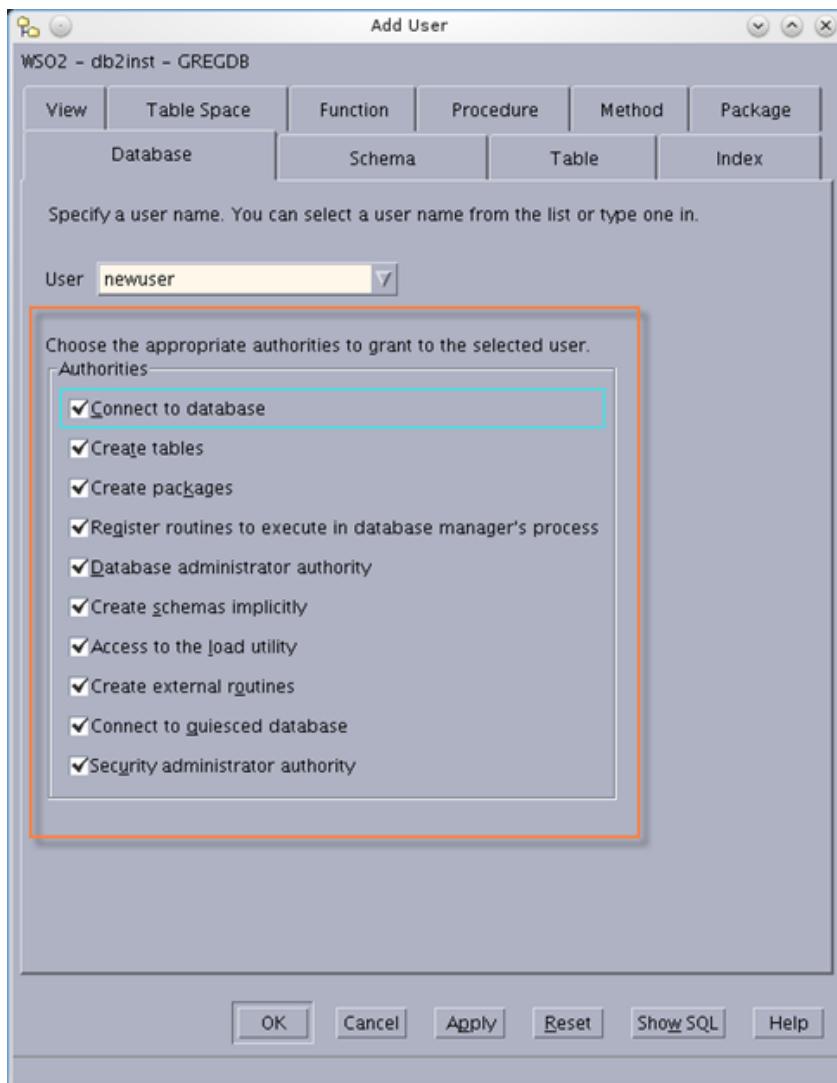
2. Right-click **All Databases** in the control center tree (inside the object browser), click **Create Database**, and then click **Standard** and follow the steps in the **Create New Database** wizard.



- Click User and Group Objects in the control center tree to create users for the newly created database.



- Give the required permissions to the newly created users.



Setting up DB2 JDBC drivers

Copy the DB2 JDBC drivers (`db2jcc.jar` and `db2jcc_license_cu.jar`) from `<DB2_HOME>/SQLLIB/java/` directory to the `<PRODUCT_HOME>/repository/components/lib/` directory.

```
user@wso2:~/sqllib/java$ cp db2jcc.jar db2jcc_license_cu.jar /home/user/wso2/greg/repository/components/lib/
user@wso2:~/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.

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.

```

<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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
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.

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 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.

```

<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>
        </configuration>
    </definition>
  </datasource>

```

The elements in the above configuration are described below:

Element	Description
---------	-------------

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.

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

You can create database tables by executing the database scripts as follows:

- Run the `ij` tool located in the `<DERBY_HOME>/bin/` directory as illustrated below:

```
client@wso2:~/dtb/db-derby-10.8.1.2-bin/bin$ ./ij
ij version 10.8
ij> ■
```

- Create the database and connect to it using the following command inside the `ij` prompt:

```
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.

- Exit from the the `ij` tool by typing the `exit` command.

```
exit;
```

- Log in to the `ij` tool with the username and password that you set in `registry.xml` and `user-mgt.xml`:

```
connect 'jdbc:derby:repository/database/WSO2CARBON_DB' user 'regadmin' password
'regadmin';
```

- 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:

```
run '<PRODUCT_HOME>/dbscripts/derby.sql';
```

Now the product is running using the embedded Apache Derby database.

- 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 configured to run using an embedded Apache Derby database.

In contrast to setting up with remote Derby, when setting up with the embedded mode, set the database driver name (the `driverClassName` element) to the value `org.apache.derby.jdbc.EmbeddedDriver` and the database URL (the `url` element) to the database directory location relative to the installation. In the above sample configuration, it is inside the `<DERBY_HOME>/WSO2_CARBON_DB/` directory.

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>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
---------	-------------

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.

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.

```
<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

You can create database tables by executing the following script(s):

1. Run the `ij` tool located in the `<DERBY_HOME>/bin/` directory.

```
client@wso2:~/dtb/db-derby-10.8.1.2-bin/bin$ ./ij
ij version 10.8
ij> █
```

2. Create the database and connect to it using the following command inside the `ij` prompt:

```
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:

```
exit;
```

4. Log in to the `ij` tool with the username and password you just used to create the database.

```
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:

```
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
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

Setting up the database

Download and install the H2 database engine in your computer.

For instructions on installing DB2 Express-C, see [H2 installation guide](#).

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 H2database, 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.

```

<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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
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.

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 the 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

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

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](#).

```
client@wso2:~/dtb$ wget -c http://www.h2database.com/h2-2011-09-11.zip
--2011-09-30 00:28:27--  http://www.h2database.com/h2-2011-09-11.zip
Resolving www.h2database.com... 80.74.147.171
Connecting to www.h2database.com|80.74.147.171|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6007851 (5.7M) [application/zip]
Saving to: "h2-2011-09-11.zip"

15% [=====] 923,304      111K/s  eta 45s
```

2. 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:

```
client@wso2:~/dtb/h2/bin$ chmod 0744 h2.sh
```

3. Run the H2 database server with the following commands:

- For Linux:
\$./h2.sh
- For Windows:
\$ h2.bat

The script starts the database engine and opens a pop-up window.

4. 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>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
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.

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**.

```

File Edit View History Bookmarks Tools Help
H2 Console - Iceweasel
http://127.0.1.1:8082/login.do?sessionid=0cd14a9efc1a047a86dea3d67b330b11
Most Visited Getting Started Latest Headlines
developerWorks : Information H2 Console WSO2 Governance Registry
Auto commit Max rows: 1000 Auto complete Normal
Run (Ctrl+Enter) Clear SQL statement:
CREATE TABLE IF NOT EXISTS REG_CLUSTER_LOCK (
    REG_LOCK_NAME VARCHAR(20),
    REG_LOCK_STATUS VARCHAR(20),
    REG_LOCKED_TIME TIMESTAMP,
    REG_TENANT_ID INTEGER DEFAULT 0,
    PRIMARY KEY (REG_LOCK_NAME)
);

CREATE TABLE IF NOT EXISTS REG_LOG (
    REG_LOG_ID INTEGER AUTO_INCREMENT,
    REG_PATH VARCHAR(2000),
    REG_USER_ID VARCHAR(31) NOT NULL,
    REG_LOGGED_TIME TIMESTAMP NOT NULL,
    REG_ACTION INTEGER NOT NULL,
    REG_ACTION_DATA VARCHAR(500),
    REG_TENANT_ID INTEGER DEFAULT 0,
    PRIMARY KEY (REG_LOG_ID, REG_TENANT_ID)
);

CREATE TABLE IF NOT EXISTS REG_PATH(
    REG_PATH_ID INTEGER NOT NULL AUTO_INCREMENT,
    REG_PATH_VALUE VARCHAR(2000) NOT NULL,
    REG_PATH_PARENT_ID INT,
    REG_TENANT_ID INTEGER DEFAULT 0,
    CONSTRAINT PK_REG_PATH PRIMARY KEY(REG_PATH_ID, REG_TENANT_ID)
);

CREATE INDEX IF NOT EXISTS REG_PATH_IND_BY_NAME ON REG_PATH(REG_PATH_VALUE, REG_TENANT_ID);
CREATE INDEX IF NOT EXISTS REG_PATH_IND_BY_PARENT_ID ON REG_PATH(REG_PATH_PARENT_ID, REG_TENANT_ID);

CREATE TABLE IF NOT EXISTS REG_CONTENT (
    REG_CONTENT_ID INTEGER NOT NULL AUTO_INCREMENT,
    REG_CONTENT_DATA LONGBLOB,
    REG_TENANT_ID INTEGER DEFAULT 0,
    CONSTRAINT PK_REG_CONTENT PRIMARY KEY(REG_CONTENT_ID, REG_TENANT_ID)
);

```

Important Commands

Displays this Help Page
Shows the Command History
Executes the current SQL statement
Disconnects from the database

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 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

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 `<INFORMIX_HOME>/etc/onconfig` file, and replace the given 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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
url	<p>The URL of the database. The default port for a DB2 instance is 50000.</p> <p>You need to add the following configuration when specifying the connection URL as shown in the example above:</p> <div style="border: 1px solid #ccc; padding: 10px; background-color: #f9f9f9;"> Add the following configuration to the connection URL when specifying it as shown in the example above: <code>CLIENT_LOCALE=en_US.utf8;DB_LOCALE=en_us.utf8;IFX_USE_STRENC=true;</code> </div>
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 the 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 a negative value to wait indefinitely.
minIdle	The minimum number of active connections that can remain idle in the pool without extra connections being created, or enter zero to create none.

testOnBorrow	The indication of whether objects will be validated before being borrowed from the pool. If an 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.

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.

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

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

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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
url	The URL of the database. Change the <IP> with the IP of the server. The best practice is to use port 1433, because you can use it in order processing services.
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.

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 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

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

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:

For instructions on installing MySQL on MAC OS, go to [Homebrew](#).

```
sudo apt-get install mysql-server mysql-client
```

2. Start the MySQL service using the following command:

```
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/co

- 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.

Do not change the datasource name WSO2_CARBON_DB in the below configurations.

```
<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>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
url	The URL of the database. The default port for MySQL is 3306
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.

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/. 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

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' ;
```

If you are using MySQL version 5.7 or later, use the following script instead:

```
mysql -u regadmin -p -Dregdb < '<PRODUCT_HOME>/dbscripts/mysql5.7.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-datasource.xml` file . Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).

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.

```
<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>
            </configuration>
        </definition>
    </datasource>
```

The elements in the above configuration are described below:

Element	Description
---------	-------------

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.

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

- To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
<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:
`COMMIT;`
4. Run the modified script using the SQL explorer as follows:

```
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
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables

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 `listner.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_OPT` 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.

```

<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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
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.

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.

```
<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

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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
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.

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 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

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
 - Changing the default WSO2_CARBON_DB datasource
 - Configuring new datasources to manage registry or user management data
- Creating database tables

Setting up the database and login role

Follow the steps below to set up a PostgreSQL database.

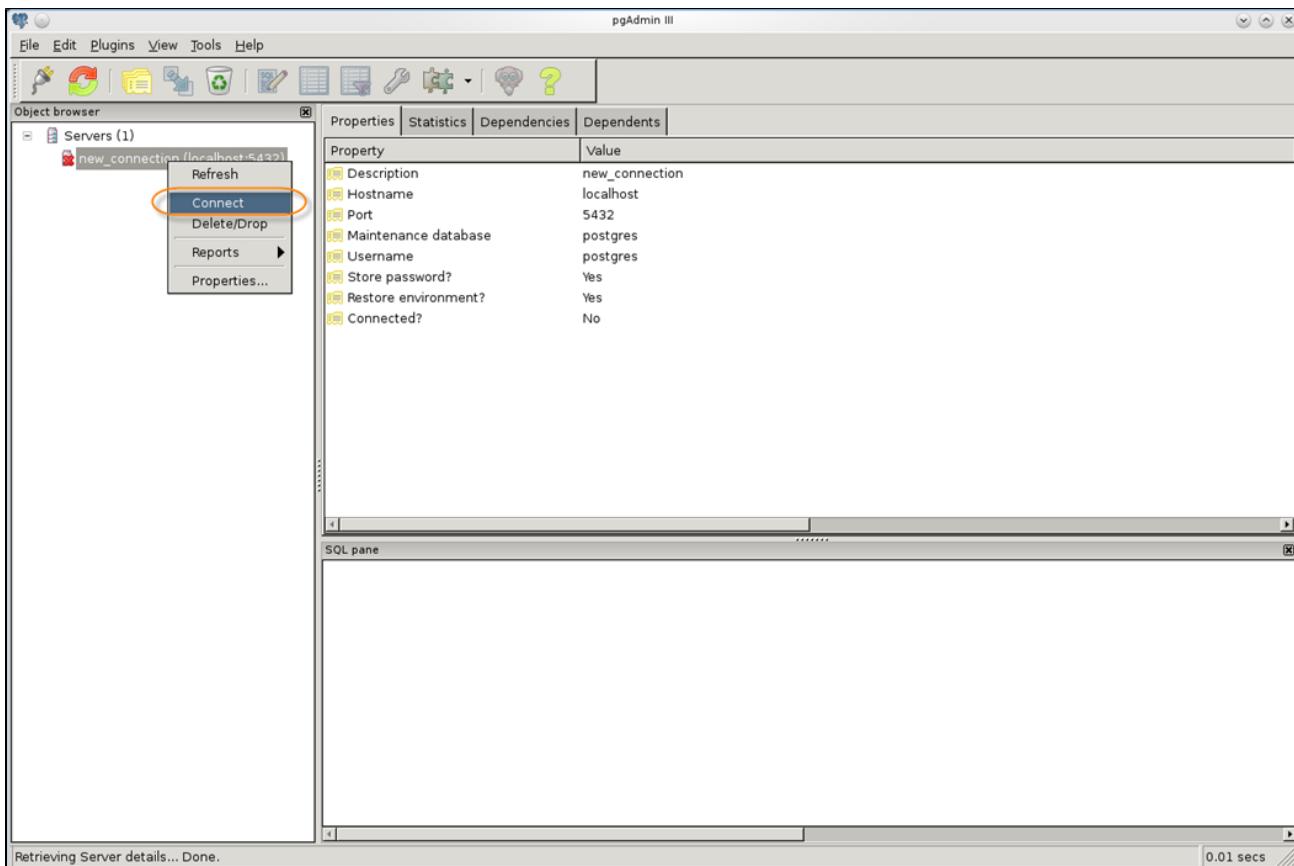
1. Install PostgreSQL on your computer as follows:

```
client@wso2:~/dtb$ sudo apt-get install postgresql
```

2. Start the PostgreSQL service using the following command:

```
client@wso2:~$ sudo /etc/init.d/postgresql start
Starting PostgreSQL 8.4 database server: main.
client@wso2:~$
```

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.

```

<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:

Element	Description
url	The URL of the database. The default port for a PostgreSQL instance is 5432.
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	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.
defaultAutoCommit	Whether to commit database changes automatically or not.
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.

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.

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

```
<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

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/>.

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](#).

For instructions on installing MariaDB on MAC OS, go to [Homebrew](#).

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>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
url	The URL of the database. The default port for MariaDB is 3306
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.

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/

```
<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:

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:

For the identity database	Use <code><DAS_HOME>/dbscripts/identity/mysql.sql</code>
For the storage database	Use <code><DAS_HOME>/dbscripts/storage/mysql.sql</code>

Configuring a Cassandra Datasource

Cassandra datasource is used to set up a connection to a Cassandra storage. In WSO2 DAS, you can create a Cassandra datasource by specifying the datasource configuration in the `<DAS_HOME>/repository/conf/datasources/ analytics-datasources.xml` file as follows.

```

<datasource>
    <name>WSO2_ANALYTICS_DS_CASSANDRA</name>
    <description>The Cassandra datasource used for analytics</description>
    <definition type="CASSANDRA">
        <configuration>
            <contactPoints>localhost</contactPoints>
            <port>9042</port>
            <username>admin</username>
            <password>admin</password>
            <clusterName>cluster1</clusterName>
            <compression>NONE</compression>
            <poolingOptions>
                <coreConnectionsPerHost
hostDistance="LOCAL">8</coreConnectionsPerHost>
                    <maxSimultaneousRequestsPerHostThreshold
hostDistance="LOCAL">1024</maxSimultaneousRequestsPerHostThreshold>
                </poolingOptions>
                <queryOptions>
                    <fetchSize>5000</fetchSize>
                    <consistencyLevel>ONE</consistencyLevel>
                    <serialConsistencyLevel>SERIAL</serialConsistencyLevel>
                </queryOptions>
                <socketOptions>
                    <keepAlive>false</keepAlive>
                    <sendBufferSize>150000</sendBufferSize>
                    <connectTimeoutMillis>12000</connectTimeoutMillis>
                    <readTimeoutMillis>12000</readTimeoutMillis>
                </socketOptions>
            </configuration>
        </definition>
    </datasource>

```

For information on the above properties, and other properties that you can specify in this datasource configuration, go to [Java Driver 2.0 for Apache Cassandra Documentation](#).

Configuring a HBase Datasource

HBase datasource is used to set up a connection to a remote HBase instance. In WSO2 DAS, you can create a

HBase datasource by specifying the datasource configurations accordingly in the <DAS_HOME>/repository/conf/datasources/analytics-datasources.xml file as shown in the example below.

```

<datasource>
    <name>WSO2_ANALYTICS_RS_DB_HBASE</name>
    <description>The datasource used for analytics file system</description>
    <jndiConfig>
        <name>jdbc/WSO2HBaseDB</name>
    </jndiConfig>
    <definition type="HBASE">
        <configuration>
            <property>
                <name>hbase.zookeeper.quorum</name>
                <value>localhost</value>
            </property>
            <property>
                <name>hbase.zookeeper.property.clientPort</name>
                <value>2181</value>
            </property>
            <property>
                <name>fs.hdfs.impl</name>
                <value>org.apache.hadoop.hdfs.DistributedFileSystem</value>
            </property>
            <property>
                <name>fs.file.impl</name>
                <value>org.apache.hadoop.fs.LocalFileSystem</value>
            </property>
        </configuration>
    </definition>
</datasource>

```

For information on the above properties, and other properties that you can specify in this datasource configuration, go to [Apache HBase Reference Guide](#).

Configuring a HDFS Datasource

HDFS datasource is used to set up a connection to a remote HDFS. In WSO2 DAS, you can create a HDFS datasource by specifying the datasource configuration in the <DAS_HOME>/repository/conf/datasources/analytics-datasources.xml file as follows.

```

<datasource>
    <name>WSO2_ANALYTICS_FS_DB_HDFS</name>
    <description>The datasource used for analytics file system</description>
    <jndiConfig>
        <name>jdbc/WSO2HDFSDB</name>
    </jndiConfig>
    <definition type="HDFS">
        <configuration>
            <property>
                <name>fs.default.name</name>
                <value>hdfs://localhost:9000</value>
            </property>
            <property>
                <name>dfs.data.dir</name>
                <value>/dfs/data</value>
            </property>
            <property>
                <name>fs.hdfs.impl</name>
                <value>org.apache.hadoop.hdfs.DistributedFileSystem</value>
            </property>
            <property>
                <name>fs.file.impl</name>
                <value>org.apache.hadoop.fs.LocalFileSystem</value>
            </property>
        </configuration>
    </definition>
</datasource>

```

For information on the above properties, and other properties that you can specify in this datasource configuration, go to [Apache Hadoop Documentation](#).

Permission enforcement for remote HDFS clients

While the HDFS cluster is being set up for use by WSO2 DAS, please ensure that the UNIX user account in all clients are present in the HDFS nodes and that all users have R/W permission to HDFS root directory.

Alternatively, you can suspend the enforcement of distributed filesystem permission enforcement through adding the following configuration to <HADOOP_HOME>/etc/hadoop/hdfs-site.xml in all HDFS nodes:

```

<property>
    <name>dfs.permissions.enabled</name>
    <value>false</value>
</property>

```

Configuring a Custom Datasource

When adding a datasource, if you select the custom datasource type, the following screen will appear:

New Data Source

New Data Source

Data Source Type* Custom

Custom Data Source Type* Table

Name*

Description

Configuration

```
<?xml version='1.0'?>
<!DOCTYPE ds SYSTEM 'http://wso2.org/dataservice'>
<ds>
<table>
<row>
<cell>1</cell>
<cell>2</cell>
</row>
</table>
</ds>
```

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.CustomTabularDataSourceReader` datasource reader implementation. If you have Data Services Server installed, for more information see the `<PRODUCT_HOME>\repository\conf\datastores\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>`:

```

<configuration>
    <customDataSourceClass>org.wso2.carbon.dataservices.core.custom.datasources.CustomQueryDataSource</customDataSourceClass>
    <customDataSourceProps>
        <property name="inmemory_datasource_schema">(Vehicles:[ID,Name,Type,Year,Color])</property>
        <property name="inmemory_datasource_records">
            <Vehicles>[{"S10_1678","Harley Davidson Ultimate Chopper","Motorcycles",1952,"Black"}, {"S10_1949","Alpine Renault 1300","Classic Cars",1952,"White"}, {"S10_2016","Moto Guzzi 1100i","Motorcycles",1996,"Black"}, {"S10_4698","Harley-Davidson Eagle Drag Bike","Motorcycles",1972,"Black"}, {"S10_4757","Alfa Romeo GTA","Classic Cars",1972,"Black"}, {"S10_4962","Lancia A Delta 16V","Classic Cars",1962,"Black"}, {"S12_1099,"Ford Mustang","Classic Cars",1968,"Black"}, {"S12_1108,"Ferrari Enzo","Classic Cars",2001,"Black"]</Vehicles>
        </property>
    </customDataSourceProps>
</configuration>

```

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.

Server Roles for C-Apps

This chapter contains the following information:

- [Introduction to Server Roles](#)
- [Adding a Server Role](#)
- [Deleting a Server Role](#)
- [Transports](#)

Introduction to Server Roles

A `ServerRoles` is a parameter that is mentioned in the `carbon.xml` file of all WSO2 Carbon based products, in the `<PRODUCT_HOME>/repository/conf` directory. Each product has a different default `ServerRoles` property as follows:

- WSO2 Application Server - "ApplicationServer"
- WSO2 Business Activity Monitor - "BusinessActivityMonitor"
- WSO2 Business Process Server - "BusinessProcessServer"
- WSO2 Business Rules Server - "BusinessRulesServer"
- WSO2 Data Analytics Server - "DataAnalyticsServer"
- WSO2 Data Services Server - "DataServicesServer"

- WSO2 Enterprise Service Bus - "EnterpriseServiceBus"
- WSO2 Governance Registry - "GovernanceRegistry"
- WSO2 Identity Server - "IdentityServer"
- WSO2 Cloud Gateway - "CloudGatewayServer"
- WSO2 API Manager - "APIManager"
- WSO2 Storage Server - "StorageServer"
- WSO2 Complex Event Processor - "ComplexEventProcessor"
- WSO2 Message Broker - "\${default.server.role}"
- WSO2 Elastic Load Balancer - "ElasticLoadBalancer"
- WSO2 User Engagement Server - "JaggeryServer"
- WSO2 Enterprise Store - "EnterpriseStore"

This property value is used for the deployment of WSO2 Carbon Applications.

What is a Carbon application

A Carbon Application, abbreviated as a C-App, is a collection of artifacts deployable on a Carbon instance. These artifacts are usually java-based or XML configurations, designed differently for each product in the Carbon platform. In a single Carbon-based solution, there can be numerous artifacts used, such as Axis2 services, data services, synapse configurations, endpoints, proxy services, mediators, registry resources, BEPL workflows etc. Usually, these artifacts are created in a development environment and then moved one by one into staging/production environments. When moving a Carbon solution from one setup to the other, the user has to manually configure these artifacts to build up the entire solution. This is a time-consuming process. Alternatively, bundling configuration files and artifacts in a C-App makes it easy for users to port their Web service based solution from one environment to another.

When a user deploys a C-App in a Carbon product, all its resources cannot be deployed in that particular product instance. To specify which can and which cannot be deployed, the `ServerRoles` property is used. When a C-App is being deployed, it reads the `ServerRoles` property from the `carbon.xml` and deploys only the resources that match the `ServerRoles` values in there. The following is an example list of C-App resources that map to default server roles.

- ApplicationServer - `foo.aar`, `jax-wx.war`
- EnterpriseServiceBus - `proxy.xml`
- BusinessProcessServer - `my_bpel.zip`
- JaggeryServer - `jaggery_app.jag`

NOTE: Carbon Applications will be renamed to Composite Applications in an upcoming release, because we will support both Carbon-based and non Carbon-based applications.

Server Roles Manager

Server roles manager is a component to manage the server roles property for WSO2 Carbon based products. Due to the functionality of the server roles manager, users do not have to manually modify the `carbon.xml` to include the server-roles related to the product feature that they have added to a Carbon product instance. The server roles manager stores both `carbon.xml` file's default product roles as well as the user/tenant specific server roles in the configuration registry. So, when a C-App is deployed in Carbon, the C-App deployer checks for auto-mentioned server roles from the registry instead of the `carbon.xml` file.

In the server roles manager, the `ServerRoles` properties are of two types:

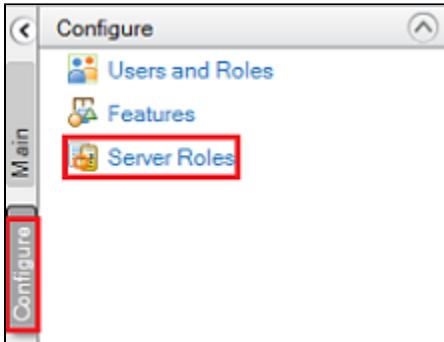
- **Default** - All the server roles picked from that particular product instance's `carbon.xml`.
- **Custom** - All other server roles added by the users.

Adding a Server Role

Follow the instructions below to add a new server role.

1. Log on to the product's Management Console and select `Server Roles` from the `Configure` menu.

For example,



2. The *Server Roles* page appears. Click on the *Add New Server Role* link.

Role Name	Type	Actions
CarbonServer	Default	Delete

Add New Server Role

3. The *Add Custom Server Role* window appears. Fill in the *Role Name* field and click the *Add* button.

Add Custom Server Role

Role Name*

Add **Cancel**

Note

You can add any textual name as a server role without special characters except underscore.

4. The new server role is added and displayed in the server roles list.

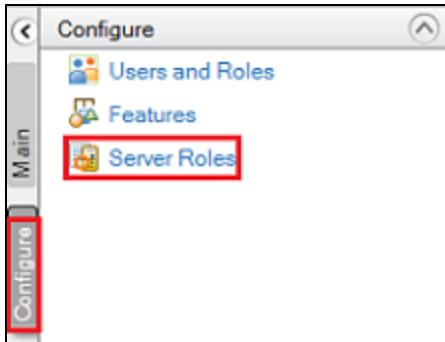
Role Name	Type	Actions
CarbonServer	Default	Delete
TestRole	Custom	Delete

Add New Server Role

Deleting a Server Role

Follow the instructions below to delete a server role.

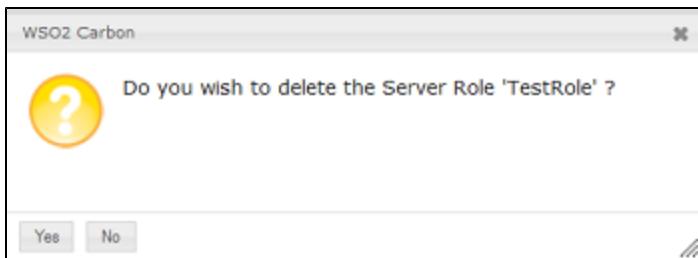
1. Log on to the product's Management Console and select *Server Roles* from the *Configure* menu.



2. The Server Roles page appears. Choose the role you want to delete and click the *Delete* link associated with it.

Role Name	Type	Actions
CarbonServer	Default	Delete
TestRole	Custom	Delete
Add New Server Role		

3. Accept the confirmation.



Note

You can't undo this operation once performed.

Note

Users can even delete a default server role. Once deleted, the server role manager will not pick up the deleted server role from the carbon.xml file, next time the server starts.

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 DAS 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](#)
- [JMS Transport](#)
- [Local Transport](#)
- [MailTo 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

Parameter Name	Description	Required	Possible Values	Default Value
----------------	-------------	----------	-----------------	---------------

port	The port number on which this transport receiver should listen for incoming messages.	Yes	A positive integer less than 65535	
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:

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"/>
```

This servlet transport implementation can be further tuned up using the following parameters.

Parameter Name	Description	Required	Possible Values	Default Value
port	The port over which this transport receiver listens for incoming messages.	Yes	A positive integer less than 65535	
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 on the proxy server. All the WSDLs generated will contain the proxy port value as the listener port.	No	A positive integer less than 65535	
maxHttpHeaderSize	The maximum size of the HTTP request and response header in bytes.	No	A positive integer	4096
maxThreads	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.	No	A positive integer	40
enableLookups	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.	No	true, false	true

disableUploadTimeout	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.	No	<i>true, false</i>	true
clientAuth	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.	No	<i>true, false, want</i>	false
maxKeepAliveRequests	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.	No	-1 or any positive integer	100
acceptCount	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.	No	A positive integer	10
compression	Use this parameter to enable content compression and save server bandwidth.	No	<i>on, off, force</i>	off
noCompressionUserAgents	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.	No	A comma separated list of regular expressions	empty string
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.catalin.a.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

Parameter Name	Description	Required	Possible Values	Default Value
PROTOCOL	The version of HTTP protocol to be used for outgoing messages.	No	<i>HTTP/1.0, HTTP/1.1</i>	HTTP/1.1
Transfer-Encoding	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.	No	<i>chunked</i>	Not Chunked
SocketTimeout	The socket timeout value in milliseconds, for out bound connections.	No	A positive integer	60000 ms
ConnectionTimeout	The connection timeout value in milliseconds, for out bound connections.	No	A positive integer	60000 ms
OmitSOAP12Action	Set this parameter to "true" if you need to disable the soapaction for SOAP 1.2 messages.	No	<i>true, false</i>	false

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.CommonsHTTPSTransportSender` 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.

Parameter Name	Description	Required	Possible Values	Default Value
sslProtocol	Transport level security protocol to be used.	No	<i>TLS, SSL</i>	TLS
keystore	Path to the keystore which should be used for encryption/decryption.	Yes	A valid file path to a keystore file	

keypass	Password to access the specified keystore.	Yes	A valid password	
---------	--------------------------------------------	-----	------------------	--

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.

JMS Transport

The Java Message Service (JMS) transport implementation also comes from the WS-Commons Transports project. All the relevant classes are packed into the `axis2-transport-jms-<version>.jar` and the following classes act as the transport receiver and the sender respectively.

- `org.apache.axis2.transport.jms.JMSListener`
- `org.apache.axis2.transport.jms.JMSSender`

The JMS transport implementation requires an active JMS server instance to be able to receive and send messages. We recommend using Apache ActiveMQ JMS server, but other implementations such as Apache Qpid and Tibco are also supported. You also need to put the client JARs for your JMS server in the product's classpath. In case of Apache ActiveMQ, you need to put the following JARs in the classpath:

- `activemq-core.jar` (For ActiveMQ 5.8.0 or above, copy `activemq-client.jar` and `activemq-broker.jar`)
- `geronimo-j2ee-management_1.0_spec-1.0.jar`

Note

For ActiveMQ 5.8.0 or above, `hawtbuf-1.2.jar` should be copied as well to `<CEP_HOME>/repository/components/lib`

These JAR files can be obtained by downloading the latest version of Apache ActiveMQ (version 5.2.0 or later is recommended). Extract the downloaded archive and find the required dependencies in the `$ACTIVEMQ_HOME/lib` directory. You need to copy these JAR files over to `<PRODUCT_HOME>/repository/components/lib` directory for the product to be able to pick them up at runtime.

Configuration parameters for JMS receiver and the sender are XML fragments that represent JMS connection factories. A typical JMS parameter configuration would look like this:

```

<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 name="transport.jms.ConnectionFactoryType">topic</parameter>
    </parameter>

```

This is a bare minimal JMS connection factory configuration which consists of four connection factory parameters. JMS connection factory parameters are described in detail below.

JMS Connection Factory Parameters

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.

Parameter Name	Description	Required	Permitted Values
java.naming.factory.initial	JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.	Yes	All
java.naming.provider.url	URL of the JNDI provider.	Yes	All
java.naming.security.principal	JNDI Username.	No	
java.naming.security.credentials	JNDI password.	No	
transport.Transactionality	Desired mode of transactionality.	No	none, required
transport.UserTxnJNDIName	JNDI name to be used to require user transaction.	No	
transport.CacheUserTxn	Whether caching for user transactions should be enabled or not.	No	true, false
transport.jms.SessionTransacted	Whether the JMS session should be transacted or not.	No	true, false
transport.jms.SessionAcknowledgement	JMS session acknowledgment mode.	No	AL, CL, DL, SE
transport.jms.ConnectionFactoryJNDIName	The JNDI name of the connection factory.	Yes	
transport.jms.ConnectionFactoryType	Type of the connection factory.	No	queue, topic
transport.jms.JMSSpecVersion	JMS API version.	No	1.1
transport.jms.UserName	The JMS connection username.	No	
transport.jms.Password	The JMS connection password.	No	
transport.jms.Destination	The JNDI name of the destination.	No	
transport.jms.DestinationType	Type of the destination.	No	queue, topic
transport.jms.DefaultReplyDestination	JNDI name of the default reply destination.	No	
transport.jms.DefaultReplyDestinationType	Type of the reply destination.	No	queue, topic
transport.jms.MessageSelector	Message selector implementation.	No	
transport.jms.SubscriptionDurable	Whether the connection factory is subscription durable or not.	No	true, false
transport.jms.DurableSubscriberClientID	The ClientId parameter when using durable subscriptions	Yes if subscription durable is turned on	true, false
transport.jms.DurableSubscriberName	Name of the durable subscriber.	Yes if subscription durable is turned on	

transport.jms.PubSubNoLocal	Whether the messages should be published by the same connection they were received.	No	true
transport.jms.CacheLevel	JMS resource cache level.	No	none
transport.jms.ReceiveTimeout	Time to wait for a JMS message during polling. Set this parameter value to a negative integer to wait indefinitely. Set to zero to prevent waiting.	No	Never
transport.jms.ConcurrentConsumers	Number of concurrent threads to be started to consume messages when polling.	No	Auto
transport.jms.MaxConcurrentConsumers	Maximum number of concurrent threads to use during polling.	No	Auto
transport.jms.IdleTaskLimit	The number of idle runs per thread before it dies out.	No	Auto
transport.jms.MaxMessagesPerTask	The maximum number of successful message receipts per thread.	No	Auto
transport.jms.InitialReconnectDuration	Initial reconnection attempts duration in milliseconds.	No	Auto
transport.jms.ReconnectProgressFactor	Factor by which the reconnection duration will be increased.	No	Auto
transport.jms.MaxReconnectDuration	Maximum reconnection duration in milliseconds.	No	

JMS transport implementation has some parameters that should be configured at service level, in other words in service XML files of individual services.

Service Level JMS Configuration Parameters

Parameter Name	Description	Required	Possible Values	Default Value
transport.jms.ConnectionFactory	Name of the JMS connection factory the service should use.	No	A name of an already defined connection factory	default
transport.jms.PublishEPR	JMS EPR to be published in the WSDL.	No	A JMS EPR	

You can find more information on JMS in WSO2 ESB documentation: [Java Message Service \(JMS\) Support](#).

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.

```
<ServerURL>local://services/&lt;/ServerURL>
```

Replace it with

```
<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:

```
<transportReceiver name="local"
class="org.wso2.carbon.core.transports.local.CarbonLocalTransportReceiver" />

<transportSender name="local"
class="org.wso2.carbon.core.transports.local.CarbonLocalTransportSender" />
```

Remove both these lines and add following line.

```
<transportSender name="local"
class="org.apache.axis2.transport.local.NonBlockingLocalTransportSender" />
```

MailTo Transport

The polling MailTo transport supports sending messages (E-Mail) over SMTP and receiving messages over POP3 or IMAP. This transport implementation is available as a module of the WS-Commons Transports project. The receiver and sender classes that should be included in the Carbon configuration to enable the MailTo transport are `org.apache.axis2.transport.mail.MailTransportListener` and `org.apache.axis2.transport.mail.MailTransportSender` respectively. The JAR consisting of the transport implementation is named `axis2-transport-mail.jar`.

The mail transport receiver should be configured at service level. That is each service configuration should explicitly state the mail transport receiver configuration. This is required to enable different services to receive mails over different mail accounts and configurations. However, transport sender is generally configured globally so that all services can share the same transport sender configuration.

Service Level Transport Receiver Parameters

The MailTo transport listener implementation can be configured by setting the parameters described in JavaMail API documentation. For IMAP related properties, see [Package Summary - IMAP](#). For POP3 properties, see [Package Summary - POP3](#). Apart from the parameters described in JavaMail API documentation, MailTo transport listener supports the following transport parameters.

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.

Parameter Name	Description	Required	Possible Values	Default Value
----------------	-------------	----------	-----------------	---------------

transport.mail.Address	The mail address from which this service should fetch incoming mails.	Yes	A valid e-mail address	
transport.mail.Folder	The mail folder in the server from which the listener should fetch incoming mails.	No	A valid mail folder name (e.g: inbox)	inbox folder if that is available or else the root folder
transport.mail.Protocol	The mail protocol to be used to receive messages.	No	<i>pop3, imap</i>	imap
transport.mail.PreserveHeaders	A comma separated list of mail header names that this receiver should preserve in all incoming messages.	No	A comma separated list	
transport.mail.RemoveHeaders	A comma separated list of mail header names that this receiver should remove from incoming messages.	No	A comma separated list	
transport.mail.ActionAfterProcess	Action to perform on the mails after processing them.	No	<i>MOVE, DELETE</i>	DELETE
transport.mail.ActionAfterFailure	Action to perform on the mails after a failure occurs while processing them.	No	<i>MOVE, DELETE</i>	DELETE
transport.mail.MoveAfterProcess	Folder to move the mails after processing them.	Required if ActionAfterProcess is MOVE	A valid mail folder name	
transport.mail.MoveAfterFailure	Folder to move the mails after encountering a failure.	Required if ActionAfterFailure is MOVE	A valid mail folder name	
transport.mail.ProcessInParallel	Whether the receiver should incoming mails in parallel or not (works only if the mail protocol supports that - for example, IMAP).	No	<i>true, false</i>	false
transport.ConcurrentPollingAllowed	Whether the receiver should poll for multiple messages concurrently.	No	<i>true, false</i>	false
transport.mail.MaxRetryCount	Maximum number of retry operations to be performed when fetching incoming mails.	Yes	A positive integer	
transport.mail.ReconnectTimeout	The reconnect timeout in milliseconds to be used when fetching incoming mails.	Yes	A positive integer	

Global Transport Sender Parameters

For a list of parameters supported by the MailTo transport sender, see [Package Summary - SMTP](#). In addition to the parameters described there, the MailTo transport sender supports the following parameters.

Parameter Name	Description	Required	Possible Values	Default Value
transport.mail.SMTPBccAddresses	If one or more e-mail addresses need to be specified as BCC addresses for outgoing mails, this parameter can be used.	No	A comma separated list of e-mail addresses	
transport.mail.Format	Format of the outgoing mail.	No	<i>Text, Multi part</i>	<i>Text</i>

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:

```
<!-- Enable TCP message -->
<transportReceiver name="tcp"
class="org.apache.axis2.transport.tcp.TCPTTransportListener">
    <parameter name="transport.tcp.port">6060</parameter>
</transportReceiver>

<transportSender name="tcp"
class="org.apache.axis2.transport.tcp.TCPTTransportSender" />
```

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

Parameter Name	Description	Required	Possible Values	Default Value
port	The port on which the TCP server should listen for incoming messages	No	A positive integer less than 65535	8000
hostname	The host name of the server to be displayed in WSDLs, etc.	No	A valid host name or an IP address	

Configuring Transports Globally

You can configure and enable transports in a service level or in a global level using either of the following methods. 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/catalina-server.xml 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.ActiveMQInitialContextFact
ory</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.ActiveMQInitialContextFact
ory</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.ActiveMQInitialContextFact
ory</parameter>
        <parameter
name="java.naming.provider.url">tcp://localhost:61616</parameter>
        <parameter
name="transport.jms.ConnectionFactoryJNDIName">QueueConnectionFactory</parameter>
    </parameter>
</transportReceiver>
```

<transportReceiver> 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:

Default HTTP/S Config in catalina-server.xml

```

<!-- optional attributes:
    proxyPort="80"-->

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
            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="ozilla, traviata"

compressableMimeType="text/html,text/javascript,application/x-javascript,application/j
avascript,application/xml,text/css,application/xslt+xml,text/xsl,image/gif,image/jpg,i
mage/jpeg" URIEncoding="UTF-8" />

<!-- optional attributes:proxyPort="443"-->

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
            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="ozilla, traviata"

compressableMimeType="text/html,text/javascript,application/x-javascript,application/j
avascript,application/xml,text/css,application/xslt+xml,text/xsl,image/gif,image/jpg,i
mage/jpeg"
            keystoreFile="${carbon.home}/repository/resources/security/wso2carbon.jks"
            keystorePass="wso2carbon"
            URIEncoding="UTF-8" />

```

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>.

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](#)) 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.

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           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.

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           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:

```
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: `ciphers="<cipher-name>,<cipher-name>"`.

```
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.

Scheduling Tasks

Task scheduling is used to invoke an operation periodically or only a specified number of times. The scheduling functionality is useful when a specific data service operation scheduled to execute is associated with an event trigger. When such a scheduled task is run, the event can be automatically fired by evaluating the event trigger criteria. For example, you can schedule a task on the `getProductQuantity` operation and set an event (e.g., send an email) if the quantity goes down to some level.

Task scheduling functionality is provided by the Data Service Tasks feature in the WSO2 feature repository. The associated identifier is `org.wso2.carbon.dataservices.task.feature.group`.

Tasks Configuration

The scheduled tasks configuration is a generic configuration, which is used by any component that requires scheduled tasks functionality. The scheduled tasks support many modes of operations and fully supports load balancing and fail-over of tasks. The tasks configuration file can be found in the `<PRODUCT_HOME>/repository/conf/etc/tasks-config.xml` file. The default configuration is shown below:

tasks-config.xml

```
<tasks-configuration xmlns:svns="http://org.wso2.securevault/configuration">

    <!--
        The currently running server mode; possible values are:-
        STANDALONE, CLUSTERED, REMOTE, AUTO.
        In AUTO mode, the server startup checks whether clustering is enabled in the
        system,
        if so, CLUSTERED mode will be used, or else, the the server mode will be
        STANDALONE.
    -->
    <taskServerMode>AUTO</taskServerMode>

    <!--
        To be used in CLUSTERED mode to notify how many servers are there in
        the task server cluster, the servers wait till this amount of servers
        are activated before the tasks are scheduled -->
    <taskServerCount>2</taskServerCount>

    <!-- The address to which the remote task server should dispatch the trigger
    messages to,
        usually this would be an endpoint to a load balancer -->
    <taskClientDispatchAddress>https://localhost:9448</taskClientDispatchAddress>

    <!-- The address of the remote task server -->
    <remoteServerAddress>https://localhost:9443</remoteServerAddress>

    <!-- The username to authenticate to the remote task server -->
    <remoteServerUsername>admin</remoteServerUsername>

    <!-- The password to authenticate to the remote task server -->
    <remoteServerPassword>admin</remoteServerPassword>

    <!-- Below contain a sample to be used when using with secure vault -->
    <!--remoteServerPassword
    svns:secretAlias="remote.task.server.password"></remoteServerPassword-->

</tasks-configuration>
```

The default values in the `tasks-config.xml` file allow the user to do minimal changes when running in both standalone and clustered modes.

The task server mode is set to `AUTO` by default, which automatically detects if clustering is enabled in the server and by default switches to clustered mode of scheduled tasks.

Task Handling Modes

There are four task handling modes available for all WSO2 Carbon servers.

- **AUTO** - This is the default task handling mode. This setting detects if clustering is enabled in the server and automatically switches to `CLUSTERED` task handling mode.
- **STANDALONE** - This mode is used when the Carbon server is used as a single installation. That is, tasks will be managed locally within the server.
- **CLUSTERED** - This mode is used when a cluster of Carbon servers are put together. With this setting, if one of the servers in the cluster fail, the tasks will be rescheduled in one of the remaining server nodes. This

requires Axis2 clustering to work.

- **REMOTE** - This mode is used when all tasks should be triggered using an independent task handling server. That is, all carbon servers using such an external task handling server should be running on **REMOTE** mode, while the task handling server can be running on **AUTO**, **STANDALONE** or **CLUSTERED** mode.

The task server count is set to two by default. This setting denotes the number of nodes in the task server cluster in the clustered mode that must be running before scheduled tasks can run, so that the scheduled tasks will be shared among the given number of nodes at startup.

For example, assume 10 tasks were saved and scheduled earlier, and for some reason later the cluster was brought down. As individual servers come back online, we do not want the first server to schedule all the tasks. Rather, we want at least two servers to come back up and share the 10 tasks.

The task clustering is based on a peer-to-peer communication mechanism. When carrying out the fail-over scenarios, it can rarely result in split-brain scenarios, where the same task can be scheduled without knowing it is already scheduled somewhere else. So the task implementors should make their best effort to make the task functionality idempotent, or come up with a mechanism to detect if the current task is already running elsewhere.

Security

The following sections explain the security aspects related to DAS.

- [Fixing Security Vulnerabilities](#)
- [Enabling Java Security Manager](#)
- [Setting up Keystores](#)

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:

```
ciphers="<cipher-name>,<cipher-name>"
```

The code below shows how a connector looks after an example configuration is done:

```

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
    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="ozilla, 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"

    ciphers="SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_AES_128_C
    BC_SHA,
    TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_SHA,SSL_RSA_WITH_3D
    ES_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.

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:

```
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.

- 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:

```
$ 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:
MD5: 54:13:FD:06:6F:C9:A6:BC:EE:DF:73:A9:88:CC:02:EC
SHA1: AE:37:2A:9E:66:86:12:68:28:88:12:A0:85:50:B1:D1:21:BD:49:52
Signature algorithm name: SHA1withRSA
Version: 3
Trust this certificate? [no]: yes
Certificate was added to keystore
```

- 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.

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
```

- 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.

- Open the startup script in the <PRODUCT_HOME>/bin folder. For Linux, it is wso2server.sh.
- Add the following system properties to the startup script and save the file:

```
-Djava.security.manager=org.wso2.carbon.bootstrap.CarbonSecurityManager \
-Djava.security.policy=$CARBON_HOME/repository/conf/sec.policy \
-Drestricted.packages=sun.,com.sun.xml.internal.ws.,com.sun.xml.internal.bind.,co
m.sun.imageio.,org.wso2.carbon. \
-Ddenied.system.properties=javax.net.ssl.trustStore,javax.net.ssl.trustStorePassw
ord,denied.system.properties \
```

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.

```
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";
};
```

Setting up Keystores

A keystore is a repository that stores the cryptographic keys and certificates that are used for various security purposes, such as encrypting sensitive information and for establishing trust between your server and outside parties that connect to your server. 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.

See the following topics for details on how keystores are used in WSO2 products and the default keystore settings with which all products are shipped:

- [Setting up keystores for WSO2 products](#)
- [Default keystore settings in WSO2 products](#)
- [Managing keystores](#)

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, xmld, xsds etc. into the registry.
- Encrypting/signing in WS-Security.

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.

See the [related links](#) for information on creating new keystores with the required certificates.

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.
- You must have the same password for both keystore and private key due to a Tomcat limitation.
- You cannot remove a service before disabling its security.

Related links

- [Configuring Keystores in WSO2 Products](#)
- [Creating New Keystores](#)
- [Managing Keystores with the UI](#)

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 it work. 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 relevant 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 go to the `<PRODUCT_HOME>/repository/conf/` directory where your product stores all configuration files.

2. Execute the following command: `grep -nr ".jks" .`

You will now get a list of configuration files and the list of keystore files that are referred to in each file. See the example 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
```

You can update the relevant configuration files as follows:

- Configuring the primary keystore
- Configuring Secure Vault for password encryption
- Configuring a keystore for SSL connections
- Configuring a keystore for Java permissions

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>
<Location>${carbon.home}/resources/security/wso2carbon.jks</Location>
<Type>JKS</Type>
<Password>wso2carbon</Password>
<KeyAlias>wso2carbon</KeyAlias>
<KeyPassword>wso2carbon</KeyPassword>
</KeyStore>

<TrustStore>
<!-- trust-store file location -->
<Location>${carbon.home}/repository/resources/security/client-truststore.jks</Location>
<!-- trust-store type (JKS/PKCS12 etc.) -->
<Type>JKS</Type>
<!-- trust-store password -->
<Password>wso2carbon</Password>
</TrustStore>
```

You need to add in the following information:

- <jks store password>
- <jks alias>
- <jks store password(same as the key password)>

Configuring Secure Vault for password encryption

All passwords in configuration files are made secure by encrypting them using cipher text. When a password is encrypted, a keystore is required for creating the decryption crypto to resolve encrypted secret values. The `secret-conf.properties` file, stored in the `<PRODUCT_HOME>/repository/conf/security/` directory is used for this purpose. Therefore, you must update this file with the relevant keystore information.

```
##KeyStores configurations
#
#keystore.identity.location=repository/resources/security/wso2carbon.jks
#keystore.identity.type=JKS
#keystore.identity.alias=wso2carbon
#keystore.identity.store.password=wso2carbon
##keystore.identity.store.secretProvider=<any implementation of
org.apache.synapse.commons.security.secret.SecretCallbackHandler>
#keystore.identity.key.password=wso2carbon
##keystore.identity.key.secretProvider=<any implementation of
org.apache.synapse.commons.security.secret.SecretCallbackHandler>
##keystore.identity.parameters=enableHostnameVerifier=false;keyStoreCertificateFilePat
h=/home/esb.cer
#
#keystore.trust.location=repository/resources/security/client-truststore.jks
#keystore.trust.type=JKS
#keystore.trust.alias=wso2carbon
#keystore.trust.store.password=wso2carbon
##keystore.trust.store.secretProvider=<any implementation of
org.apache.synapse.commons.security.secret.SecretCallbackHandler>
```

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.

```
keystoreFile="${carbon.home}/repository/resources/security/wso2carbon.jks"
keystorePass="wso2carbon"
```

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.

```
keystore "file:${user.dir}/repository/resources/security/wso2carbon.jks", "JKS";
```

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:

```
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:

```
keytool -genkey -alias certalias -keyalg RSA -keysize 2048 -keystore newkeystore.jks -dname "CN=<testdomain.org>,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass mypassword -keypass mypassword
```

This command will create a keystore with the following details:

- Keystore name: newkeystore.jks
- Alias of public certificate: certalias
- Keystore password: mypassword
- Private key password: mypassword (this is required to be the same as keystore password)

Note that if you did not specify values for the '-keypass' and the '-storepass' in the above command, you will be asked to give a value for the '-storepass' (password of the keystore). As a best practice, use a password generator to generate a strong password. You will then be asked to enter a value for -keypass. Click **Enter**, because we need the same password for both the keystore and the key. Also, if you did not specify values for -dname, you will be asked to provide those details individually.

- Open the <PRODUCT_HOME>/repository/resources/security/ directory and check if the new keystore file is created. Make a backup of it and move it to a secure location. This is important as it is the only place with our private key.

Step 2: Creating CA-signed certificates for public key

Now we have a .jks file. This keystore (.jks file) can be used to generate a certificate signing request (CSR). This CSR file must be certified by a certificate authority or certification authority (CA), which is an entity that issues digital certificates. These certificates can certify the ownership of a public key.

- 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.

- 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.

- 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 , COMODORSADomainValidationSecureServerCA.crt SSL Certificate signed by CA: test_sampleapp_org.crt

Step 3: Importing CA-signed certificates to keystore

- 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 COMODORSADomainValidationSecureServerCA.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.

- 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 example.

To add the public key of the signed certificate to the client trust store:

- Get a copy of the `client-truststore.jks` file from the `<PRODUCT_HOME>/repository/resources/security/` directory.
- Export the public key from your `.jks` file using the following command.

```
keytool -export -alias certalias -keystore newkeystore.jks -file <public key name>.pem
```

- Import the public key you extracted in the previous step to the `client-truststore.jks` file using the following command.

```
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

If you are integrating WSO2 DAS with WSO2 CEP, uncomment the `<thriftAgentConfiguration>` property and change the default password values of it accordingly in the `<DAS_HOME>/repository/conf/data-bridge/thrift-agent-config.xml` file as shown below.

```
<thriftAgentConfiguration xmlns="http://wso2.org/carbon/databridge/agent/thrift">
<!--<trustStore>
    .../wso2cep-1.0.0/repository/resources/security/client-truststore.jks
</trustStore>
<trustStorePassword>wso2carbon</trustStorePassword>-->
<trustStorePassword>wso2carbon</trustStorePassword>-->
```

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. See the [related topics](#) for more information.

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

KeyStore File	
KeyStore File*	<input type="button" value="Choose File"/> no file selected
KeyStore Password*	*****
Provider	admin
KeyStore Type	JKS
<input type="button" value="Next >"/> <input type="button" value="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

Search		
Enter Keystore name pattern (* for all)		*
Name	Type	Actions
wso2carbon.jks	JKS	Import Cert View Delete

[Add New KeyStore](#)

4. Click **View** in the list of actions. The **View Key Store** screen shows information about the available certificates.

Available Certificates

Search							
Enter Certificate alias pattern (* for all)							
Alias	IssuerDN	NotAfter	NotBefore	SerialNumber	SubjectDN	Version	Actions
wso2carbon.cert	CN=wso2carbon, OU=None, L=Seattle, ST=Washington, O=WSO2, C=LK	25/09/2282	11/12/2008	1228997318	CN=wso2carbon, OU=None, L=Seattle, ST=Washington, O=WSO2, C=LK	1	
entrustclientca	CN=Entrust.net Client Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/Client_CA_Info/CPS incorp. by ref. limits lab., O=Entrust.net, C=US	13/10/2019	13/10/1999	939758062	CN=Entrust.net Client Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/Client_CA_Info/CPS incorp. by ref. limits lab., O=Entrust.net, C=US	3	
verisignclass3g2ca	OU=VeriSign Trust Network, OU=(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 3 Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US	02/08/2028	18/05/1998	167285380242319648451154478808036881606	OU=VeriSign Trust Network, OU=(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 3 Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US	1	
thawtepersonalbasicca	EMAILADDRESS=personal-basic@thawte.com, CN=Thawte Personal Basic CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA	01/01/2021	01/01/1996	0	EMAILADDRESS=personal-basic@thawte.com, CN=Thawte Personal Basic CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA	3	
verisignclass2g3ca	CN=VeriSign Class 2 Public Primary Certification Authority - G3, OU=(c) 1999 VeriSign, Inc. - For authorized use only", OU=VeriSign Trust Network, O=VeriSign, Inc., C=US	17/07/2036	01/10/1999	129520775995541613599859419027715677050	CN=VeriSign Class 2 Public Primary Certification Authority - G3, OU=(c) 1999 VeriSign, Inc. - For authorized use only", OU=VeriSign Trust Network, O=VeriSign, Inc., C=US	1	
thawtepersonalpremiumca	EMAILADDRESS=personal-premium@thawte.com, CN=Thawte Personal Premium CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA	01/01/2021	01/01/1996	0	EMAILADDRESS=personal-premium@thawte.com, CN=Thawte Personal Premium CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA	3	
valicertclass2ca	EMAILADDRESS=info@valicert.com, CN=http://www.valicert.com/, OU=ValiCert Class 2 Policy Validation Authority, O=ValiCert, Inc., L=ValiCert Validation Network	26/06/2019	26/06/1999	1	EMAILADDRESS=info@valicert.com, CN=http://www.valicert.com/, OU=ValiCert Class 2 Policy Validation Authority, O=ValiCert, Inc., L=ValiCert Validation Network	1	
entrustsslc	CN=Entrust.net Secure Server Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/CPS incorp. by ref. (limits lab.), O=Entrust.net, C=US	25/05/2019	25/05/1999	927650371	CN=Entrust.net Secure Server Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/CPS incorp. by ref. (limits lab.), O=Entrust.net, C=US	3	
equifaxsecurebusinessca2	OU=Equifax Secure eBusiness CA-2, O=Equifax Secure, C=US	23/06/2019	23/06/1999	930140085	OU=Equifax Secure eBusiness CA-2, O=Equifax Secure, C=US	3	
equifaxsecurebusinessca1	OU=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US	21/06/2020	21/06/1999	4	OU=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US	3	

<< first <Prev 1 2 3 ... Next > last >

It also displays information about private key certificates:

Certificate of the Private Key

Alias	IssuerDN	NotAfter	NotBefore	SerialNumber	SubjectDN	Version
wso2carbon	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	13/02/2035	19/02/2010	1266562946	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	3
Import Cert Finish						

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.

verisignclass3ca
entrustgssica
geotrustglobalca
verisignclass1g2ca
mycert

Related topics**Connecting a DAS Instance to an Existing External Apache Spark Cluster**

This section explains how to start a DAS instance and connect it to an Apache Spark cluster in a location different to that DAS instance.

Prerequisites

You are required to have access to an existing external Apache Spark cluster. For more information about configuring an Apache Spark cluster, see [Apache Spark Documentation - Cluster Mode Overview](#).

Configuring DAS to connect to an external Apache Spark cluster

1. Download WSO2 DAS from [here](#). Unzip the file you downloaded in your node, as well in each node of the external Apache Spark cluster.
2. Set up MySQL as follows in your node, as well in each node of the external Apache Spark cluster. WSO2 DAS is configured with MySQL in this scenario because the datasource used needs to be of a type that can be accessed by the external Apache Spark cluster.
 - a. Download and install [MySQL Server](#).
 - b. Download the [MySQL JDBC driver](#).
 - c. Unzip the downloaded MySQL driver zipped archive, and copy the MySQL JDBC driver JAR (`mysql-connector-java-x.x.xx-bin.jar`) into the `<Das_Home>/repository/components/lib` directory of all the nodes in the cluster.
 - d. Enter the following command in a terminal/command window, where `username` is the username you want to use to access the databases.
`mysql -u username -p`
 - e. When prompted, specify the password that will be used to access the databases with the `username` you specified.
 - f. Create two databases named `userdb` and `regdb`.

About using MySQL in different operating systems

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 following is how your database creation command should look.

```
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 following is how your database creation command should look.

```
mysql> create database <DATABASE_NAME>;
```

- g. Execute the following script for the two databases you created in the previous step.
`mysql> source <Das_Home>/dbscripts/mysql.sql;`
 ↴ [Click here to view the commands for performing steps 6 and 7](#)

```
mysql> create database userdb;
mysql> use userdb;
mysql> source <DAS_HOME>/dbscripts/mysql.sql;
mysql> grant all on userdb.* TO username@localhost identified by
"password";

mysql> create database regdb;
mysql> use regdb;
mysql> source <DAS_HOME>/dbscripts/mysql.sql;
mysql> grant all on regdb.* TO username@localhost identified by
"password";
```

- h. Create the following databases in MySQL.
- WSO2_ANALYTICS_EVENT_STORE_DB
 - WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB
3. Point to WSO2_ANALYTICS_EVENT_STORE_DB and WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB in the <DAS_HOME>/repository/conf/datasources/analytics-datasources.xml file as shown below. This configuration should be done in your node, as well in each node of the external Apache Spark cluster.

```

<datasources-configuration>
    <providers>

        <provider>org.wso2.carbon.ndatasource.rdbms.RDBMSDataSourceReader</provider>
    </providers>

    <datasources>
        <datasource>
            <name>WSO2_ANALYTICS_EVENT_STORE_DB</name>
            <description>The datasource used for analytics record store</description>
            <definition type="RDBMS">
                <configuration>
                    <url>jdbc:mysql://[MySQL DB url]:[port]/WSO2_ANALYTICS_EVENT_STORE_DB</url>
                    <username>[username]</username>
                    <password>[password]</password>
                    <driverClassName>com.mysql.jdbc.Driver</driverClassName>
                    <maxActive>50</maxActive>
                    <maxWait>60000</maxWait>
                    <testOnBorrow>true</testOnBorrow>
                    <validationQuery>SELECT 1</validationQuery>
                    <validationInterval>30000</validationInterval>
                    <defaultAutoCommit>false</defaultAutoCommit>
                </configuration>
            </definition>
        </datasource>
        <datasource>
            <name>WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB</name>
            <description>The datasource used for analytics record store</description>
            <definition type="RDBMS">
                <configuration>
                    <url>jdbc:mysql://[MySQL DB url]:[port]/WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB</url>
                    <username>[username]</username>
                    <password>[password]</password>
                    <driverClassName>com.mysql.jdbc.Driver</driverClassName>
                    <maxActive>50</maxActive>
                    <maxWait>60000</maxWait>
                    <testOnBorrow>true</testOnBorrow>
                    <validationQuery>SELECT 1</validationQuery>
                    <validationInterval>30000</validationInterval>
                    <defaultAutoCommit>false</defaultAutoCommit>
                </configuration>
            </definition>
        </datasource>
    </datasources>
</datasources-configuration>

```

For more information, see [Datasources](#).

4. Create a symbolic link pointing to the <DAS_HOME> in your DAS node as well as in each node of the external Apache Spark cluster. The path in which this symbolic link is located should be the same for each node. You can use a command similar to the following command in order to create the symbolic link (change the location specified as required).

```
sudo ln -s /home/centos/das/wso2das-3.0.0-SNAPSHOT das_symlink
```

This creates a symbolic link in the `mnt/DAS` location.

5. Do the following in the <DAS_home>/repository/conf/analytics/spark/spark-defaults.conf file in your DAS node as well as all the nodes of the external Apache Spark cluster.
 - a. Set the carbon.spark.master property to cluster.
 - b. Add the carbon.das.symbolic.link property and enter the symbolic link you created in the previous step as the value.
 6. Start the external Apache Spark cluster if it is not already started.
 7. Start the DAS instance It will connect to the external Apache Spark cluster at start up.
 8. Go to external-spark-node-IP:4040 and check whether you can see the following web page.

The class paths listed in the `spark.driver.extraClassPath` should include the following symbolic link location.

(mnt/das/das_symlink)

Storing Index Data

Index data in WSO2 DAS is stored in a local file system. All index data are partitioned into units known as shards. These shards can be viewed in the `<Das_Home>/repository/data/index_data` directory where there is a sub directory for each available shard.

Configuring shards

Shards that exist in the local file system can be managed by configuring the following parameters in the `<Das_Home>/repository/conf/analytics/analytics-config.xml` file.

Parameter	Description	Default
indexReplicationFactor	The number of index data replicas that should be saved in the system. In a high availability deployment, this at least one replica should be saved.	1

shardCount	The number of index shards that are allowed to exist in the local file system at a given time. The number specified should be higher than the number of indexing nodes in the DAS cluster. The ideal number can be calculated as follows. number of indexing nodes * [CPU cores used for indexing per node]	6
shardIndexRecordBatchSize	The amount of index data to be processed by a shard index worker at a given time. This is expressed in bytes. The minimum amount should be 1000.	20971520
shardIndexWorkerInterval	The time interval during which a shard index processing worker can be inactive while processing operations are taking place, expressed in milliseconds. This parameter, together with the shardIndexRecordBatchSize parameter can be used to increase the final index batched data amount the an index worker processes at a given time. A higher batch data amount usually results in a higher throughput. However, it can also increase the latency of record insertion to indexing. The minimum value is 10, and the maximum value is 60000 (1 minute).	1500

Allocating shards in a clustered deployment

In a WSO2 DAS cluster, the available shards are equally distributed among all the indexing nodes (i.e. nodes for which indexing is enabled). e.g., if the cluster has 3 indexing nodes and 6 shards, each indexing node is assigned two shards (i.e unless replication is enabled). When a new indexing node joins a cluster, the existing shard allocations change in order to assign some of the shards to the new node.

If you do not want a new node to operate as an indexing node, you should disable indexing at the time the node is started, using the following setting.

```
disableIndexing=true
```

If you want to stop an existing node operating as an indexing node, you should restart it with the same setting. As a result, the existing shard allocation in the indexing cluster changes in order to reallocate the shards of the quitting node to other indexing nodes.

Mistakenly started indexing nodes

If you start a node as an indexing node by mistake, it changes global configurations and these changes need to be reverted manually. If the replication factor is equal to or greater than 1, you can still query and get the required data even if this node is inactive by following the procedure below.

1. Restart the node as a non-indexing node (i.e. by setting the `disableIndexing=true` property at the time the node is restarted).
2. If you want to clear the index data stored in the node, delete them from `<Das_Home>/repository/data/indexing_data` directory.
3. If you want to use the node in another server profile, restart the node in the required profile.

- When you restart an indexing node as a non-indexing node, you should also restart the other indexing servers for them to get the indexing updates of the node that stopped operating as an indexing node.
- If you start an indexing server by mistake, it changes the global configurations. You need to make

sure that the shard allocations are correct before proceeding.

When an indexing node is restarted as a non indexing node, the indexing data stored in it is not automatically removed. You can remove it if required from the <DAS-HOME>/repository/data/indexing_data directory.

Allocating shards manually

Shards can be configured manually in the <DAS_HOME>/repository/conf/analytics/local-shard-allocation-config.conf file.

There are three modes when configuring the local shard allocations of a node.

Mode	Description
NORMAL	The indexing data for a shard is stored in the node to which the shard is assigned.
INIT	If you restart the server after adding a shard in the INIT mode, that shard would be re-indexed in that node. e.g., If the existing shard allocations are as follows, and you add the line 4, INIT and restart the server in order to reindex the data for shard 4. After the data is reindexed, the mode is changed to NORMAL. <div style="border: 1px solid black; padding: 10px; width: fit-content;"> 1, NORMAL 2, NORMAL </div>
RESTORE	This mode allows you to copy index data to a local node in order to let that node use it. e.g., If you copy index data for shard 5, add the line 5, RESTORE to the following shard allocation, and then restart the server, the node allocates the 5th shard to that node (which is then used to search and index). <div style="border: 1px solid black; padding: 10px; width: fit-content;"> 1, NORMAL 2, NORMAL </div>

Related Links

- [Configuring Data Persistence](#) - For an introduction about persisting data (both records and index data) in WSO2 DAS.
- [Configuring Indexes](#) - For detailed information about indexing in WSO2 DAS.

Configuring Single Sign-On for WSO2 DAS

SSO (Single Sign-On) allows you to be authenticated to access one application, and gain access to multiple other applications without having to repeatedly provide your credentials for authentication purposes.

To configure SSO for the WSO2 DAS Portal and the WSO2 DAS Management Console, follow the instructions given below:

- [Prerequisites](#)

- Step 1: Share a user store between WSO2 DAS and WSO2 IS
- Step 2: Mount and share a registry between WSO2 DAS and WSO2 IS
- Step 3: Create a service provider for WSO2 DAS Management Console
- Step 4: Create a service provider for WSO2 DAS Portal
- Step 5: Update the SAML2SSOAuthenticator configuration
- Step 6: Update the authentication configuration

Prerequisites

In order to configure SSO for WSO2 DAS, WSO2 IS should be downloaded and installed.

To download this product, go to the [WSO2 Identity Server home page](#).

For detailed instructions, see [WSO2 Identity Server Documentation - Installation Prerequisites](#).

Step 1: Share a user store between WSO2 DAS and WSO2 IS

Follow the procedure below to share a user store between WSO2 DAS and WSO2 IS.

1. Create a new database named `DAS_UM_DB` in MYSQL server.
2. Create tables inside the `DAS_UM_DB` database by executing the script in the `<DAS_HOME>/dbscripts/mysql.sql`.
3. Define a datasource as follows in the `<DAS_HOME>/repository/conf/datasources/master-datasources.xml` file. This allows you to connect to the `DAS_UM_DB` in order to share the user store.

```

<datasource>
    <name>WSO2UM_DB</name>
    <description>The datasource used for user manager database</description>
    <jndiConfig>
        <name>jdbc/WSO2UM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/DAS_UM_DB</url>
            <username>username</username>
            <password>password</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>

```

4. Add the same datasource configuration given in the previous step to the `<IS_HOME>/repository/conf/datasources/master-datasources.xml` file.
5. Download the MySQL database driver and copy it to the `<IS_HOME>/repository/components/lib` and the `<DAS_HOME>/repository/components/lib` directories.
6. Update the `<DAS_HOME>/repository/conf/user-mgt.xml` file with the `jndiConfig` name added in step3 (i.e., `jdbc/WSO2UM_DB`) as shown below. Do the same update in the `<IS_HOME>/repository/conf/user-mgt.xml` file.

```
<configuration>
  ...
    <Property name="dataSource">jdbc/WSO2UM_DB</Property>
</configuration>
```

7. Copy the following JDBC user store configuration that can be found in the <DAS_HOME>/repository/conf/user-mgt.xml file to the <IS_HOME>/repository/conf/user-mgt.xml file. Remove the LDAP user store configuration available by default in the <IS_HOME>/repository/conf/user-mgt.xml file.

```
<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="ReadGroups">true</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>
```

8. Restart both WSO2 DAS and WSO2 IS servers.

Step 2: Mount and share a registry between WSO2 DAS and WSO2 IS

Follow the procedure below to share a registry between WSO2 DAS and WSO2 IS.

1. Create a new database named DAS_REG_DB in the MySQL server.
2. Create tables in the database you created by executing the script in the <DAS_HOME>/dbscripts/mysql.sql directory.
3. Define a datasource as follows in the <DAS_HOME>/repository/conf/datasources/master-datasources.xml file. This allows you to connect to the DAS_REG_DB that you previously created.

```

<datasource>
    <name>WSO2REG_DB</name>
    <description>The datasource used for registry database</description>
    <jndiConfig>
        <name>jdbc/WSO2REG_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/DAS_REG_DB</url>
            <username>username</username>
            <password>password</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>

```

4. Add the same datasource configuration provided in the above step to the <IS_HOME>/repository/conf/datasources/master-datasources.xml file.
5. Download the MySQL database driver from [here](#) and copy it to both the <IS_HOME>/repository/components/lib and the <DAS_HOME>/repository/components/lib directories.
6. Create the registry mounts by adding the following configuration to both the <DAS_HOME>/repository/conf/registry.xml file and the <IS_HOME>/repository/conf/registry.xml file.

```

<dbConfig name="govregistry">
    <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
    <id>gov</id>
    <dbConfig>govregistry</dbConfig>
    <readOnly>false</readOnly>
    <enableCache>true</enableCache>
    <registryRoot>/</registryRoot>
</remoteInstance>

<mount path="/_system/governance" overwrite="true">
    <instanceId>gov</instanceId>
    <targetPath>/_system/governance</targetPath>
</mount>

<mount path="/_system/config" overwrite="true">
    <instanceId>gov</instanceId>
    <targetPath>/_system/config</targetPath>
</mount>

```

7. Restart both the WSO2 DAS server and the WSO2 IS server.

Step 3: Create a service provider for WSO2 DAS Management Console

Follow the procedure below to configure a service provider for the WSO2 DAS Management Console. A service provider represents an external application from which a service is obtained. In this scenario, the service provider you create provides an authentication service. For more information about service providers, see [WSO2 Identity Server Documentation - Configuring a Service Provider](#).

1. Access the WSO2 IS Management Console using the following URL, and log in using your credentials.
`https://<IS_HOST>:<IS_PORT>/carbon/`
2. Click **Main => Service Providers => Add** to open the **Add New Service Provider** page. Enter the following details, and click **Register** to register a new service provider.

Parameter	Value
Service Provider Name	DAS_SSO_CARBON_SERVER
Description	Single Sign-On for DAS Management Console.
SaaS Application	Select this check box. This makes the web application represented by the service provider accessible for all the users of all the tenants connected to the service provider.

3. In the **Service Providers** page, expand the **Inbound Authentication Configuration** section. Then expand the **SAML2 Web SSO Configuration** section and click **Configure**. This opens the **Register New Service Provider** page.
4. Enter information as follows.

Parameter	Value
Issuer	carbonServer
Assertion Consumer URL	<code>https://<DAS_URL>:<DAS_PORT>/acs</code>
Enable Response Signing	Select this check box.
Enable Single Logout	Select this check box.

5. Expand the **Local and Outbound Authentication Configuration** section, and select the **Use tenant domain in local subject identifier** check box. This appends the tenant domain to the local subject identifier and allows the tenant domains of the application users to be identified.
6. Click **Update** to save these changes. A message appears to confirm that the service provider is successfully added.

Step 4: Create a service provider for WSO2 DAS Portal

Follow the procedure below to configure a service provider for the WSO2 DAS Portal. A service provider represents an external application from which a service is obtained. In this scenario, the service provider you create provides an authentication service. For more information about service providers, see [WSO2 Identity Server Documentation - Configuring a Service Provider](#).

1. Log into the WSO2 IS Management Console if you are not already logged in.
2. Click **Main => Service Providers => Add** to open the **Add New Service Provider** page. Enter the following details, and click **Register** to register a new service provider.

Parameter	Value
-----------	-------

Service Provider Name	DAS_SSO_PORTAL
Description	Single Sign-On for DAS Portal.
SaaS Application	<p>Select this check box.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> This makes the web application represented by the service provider accessible for all the users of all the tenants connected to the service provider. </div>

3. In the **Service Providers** page, expand the **Inbound Authentication Configuration** section. Then expand the **SAML2 Web SSO Configuration** section and click **Configure**. This opens the **Register New Service Provider** page.
4. Enter information as follows.

Parameter	Value
Issuer	portal
Assertion Consumer URL	<code>https://<DAS_URL>:<DAS_PORT>/portal/acs</code>
Enable Response Signing	Select this check box.
Enable Single Logout	Select this check box.
Enable Audience Restriction	Select this check box and add the following two audiences. <ul style="list-style-type: none"> Token endpoint url (eg: <code>https://<IDP_URL>:<IDP_PORT>/oauth2/token</code>) Management console issuer name (i.e. carbonServer)
Enable Recipient Validation	Select this check box and add the token endpoint URL as a recipient (e.g., <code>https://<IDP_URL>:<IDP_PORT>/oauth2/token</code>).

5. Expand the **Local and Outbound Authentication Configuration** section, and select the **Use tenant domain in local subject identifier** check box. This appends the tenant domain to the local subject identifier. This allows the tenant domains of the application users to be identified.
6. Click **Update** to save these changes. A message appears to confirm that the service provider is successfully added.

Step 5: Update the SAML2SSOAuthenticator configuration

This step is carried out in order to connect the service provider you created as an authenticator for WSO2 DAS. Follow the procedure below to update the `authenticators.xml` file with the information required.

1. Open the `<DAS_HOME>/repository/conf/security/authenticators.xml` file.
2. Enable the required authenticator by setting `disabled="false"` as shown in the example below.

```
<Authenticator name="SAML2SSOAuthenticator" disabled="true">
```

3. Configure the following parameters for the same authenticator that you enabled in the previous step.

Parameter	Value
ServiceProviderID	carbonServer This should be the ID of the service provider you created in Step 3.
IdentityProviderSSOServiceURL	<a href="https://<IDP_URL>:<IDP_PORT>/samlss">https://<IDP_URL>:<IDP_PORT>/samlss
AssertionConsumerServiceURL	<a href="https://<DAS_URL>:<DAS_PORT>/acs">https://<DAS_URL>:<DAS_PORT>/acs

- Save the changes.

Step 6: Update the authentication configuration

Follow the procedure below in order to specify SSO as the currently active authentication method for WSO2 DAS.

- Open the <DAS_HOME>/repository/deployment/server/jaggeryapps/portal/configs/designer.json file.
- Update the properties in the "authentication" configuration as follows.

Property	Value
activeMethod	sso
issuer	portal
identityProviderURL	<a href="https://<IDP_URL>:<IDP_PORT>/samlss">https://<IDP_URL>:<IDP_PORT>/samlss
acs	<a href="https://<DAS_URL>:<DAS_PORT>/portal/acs">https://<DAS_URL>:<DAS_PORT>/portal/acs

```

"activeMethod": "basic",
"methods": {
    "sso": {
        "attributes": {
            "issuer": "portal",
            "identityProviderURL": "https://localhost:9443/samlss",
            "responseSigningEnabled": true,
            "validateAssertionValidityPeriod": true,
            "validateAudienceRestriction": true,
            "assertionSigningEnabled": true,
            "acs": "https://localhost:9444/portal/acs",
            "identityAlias": "wso2carbon",
            "defaultNameIDPolicy": "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified",
            "useTenantKey": false,
            "isPassive": false
        }
    },
}
,
```

- Save the changes.

Restart the WSO2 DAS server after you carry out all the steps given above.

Deployment and Clustering

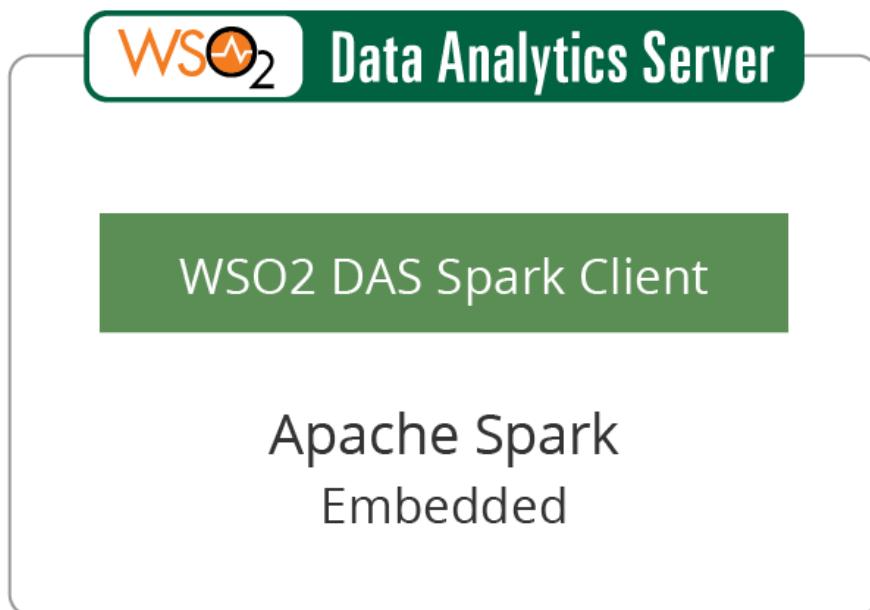
The topics in this section provide instructions on how to deploy and configure clustering for WSO2 Data Analytics Server (DAS). The configurations change depending on the deployment pattern you choose. The following are the possible deployment patterns available for WSO2 DAS.

Deployment patterns

The following are the main deployment patterns available. Note that not all of these are patterns recommended for production environments. The reasoning behind the recommendation is provided for each pattern. See [Spark Deployment Patterns](#) for deployment patterns related to the embedded Apache Spark that comes with the WSO2 Data Analytics Server.

Single node deployment

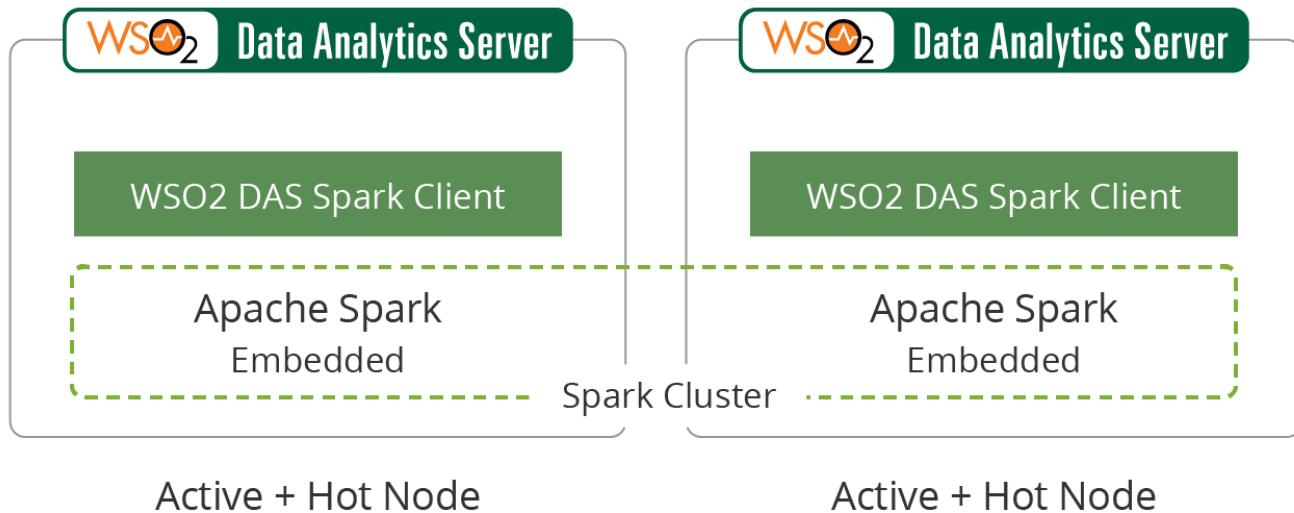
WSO2 Data Analytics Server can be deployed as a single instance in a server. This is not recommended in a typical production environment as it does not offer high availability.



High availability deployment

WSO2 Data Analytics Server can be deployed with high availability by having a minimum of two instances in the production environment. In this instance, both nodes are active and hot. What this means is that if one node fails the other can take over without any hesitation and they both have the ability to serve requests. The embedded Apache Spark in the WSO2 DAS instance works as though it is clustered with the embedded Apache Spark in the other WSO2 DAS instance.

Note: You can have more than two nodes if you want to ensure that there is more availability in case of failure. However, when there are too many nodes in the cluster, there is an impact on the performance of the cluster. This occurs because it may take some time to locate members of the cluster if there are many.



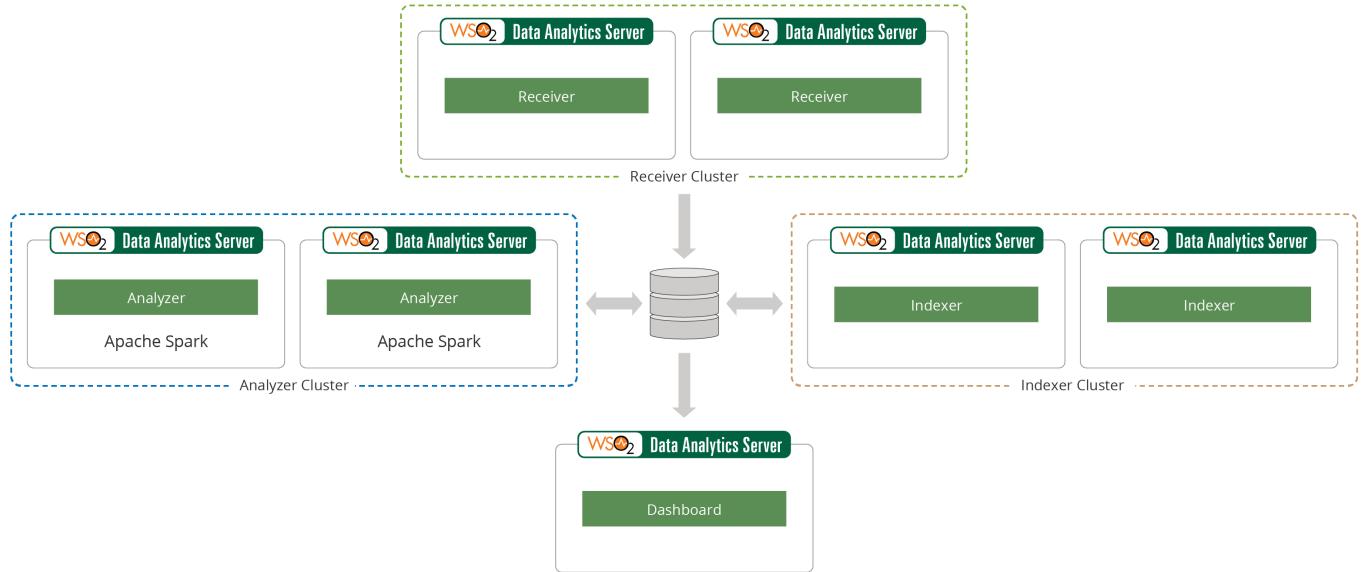
Fully distributed deployment

WSO2 Data Analytics Server can be deployed by distributing the tasks to various instances and clustering these instances for high availability.

Note: You can have more than two nodes if you want to ensure that there is more availability in case of failure. However, when there are too many nodes in the cluster, there is an impact on the performance of the cluster. This occurs because it may take some time to locate members of the cluster if there are many.

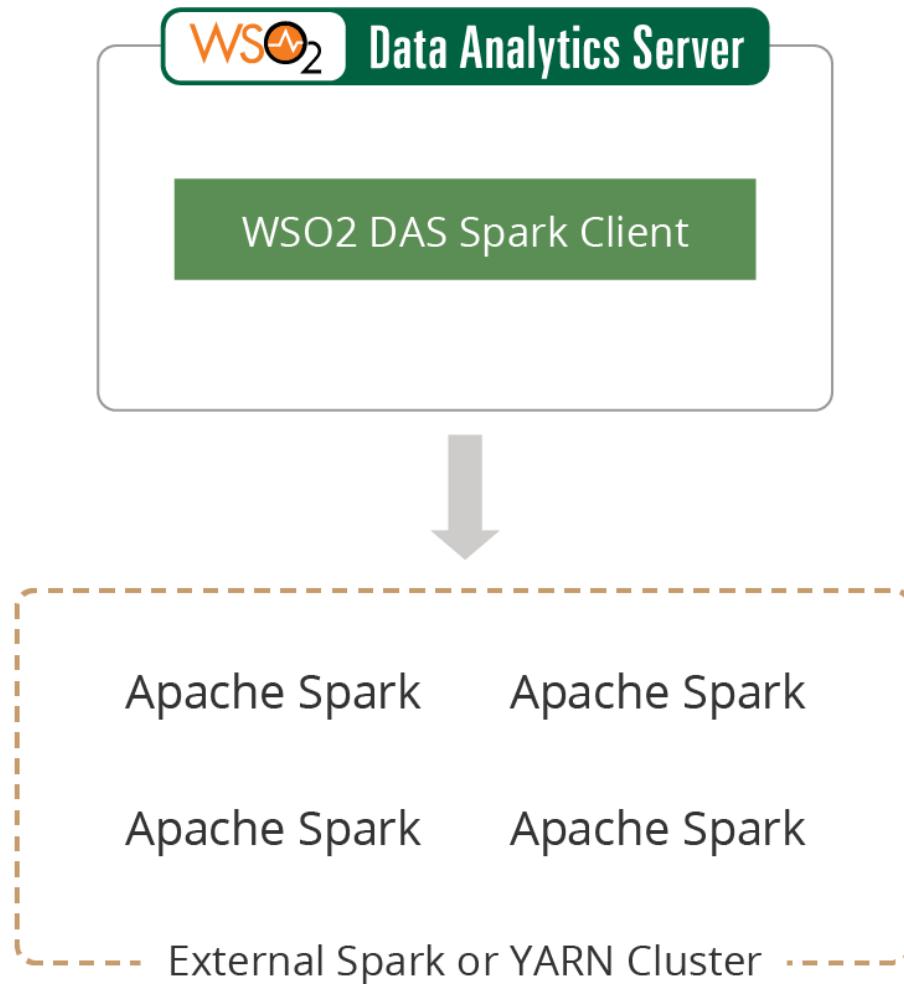
The following table describes these distributed components.

Distributed component	Minimum number of nodes	Description
Receiver nodes	2	For data analytics to happen, it is necessary to first collect the relevant data you require. DAS provides data agents that capture information on the messages flowing through the WSO2 ESB, WSO2 Application Server, and other products that use the DAS data publisher. The information is obtained by the data receivers and is then stored in a datastore, where it is optimized for analysis. The receiver nodes are used to obtain this data from the data agents.
Indexer nodes	2	A background indexing process fetches the data from the datastore and does the indexing operations. These operations are handled by the indexer nodes in a fully distributed, highly available system.
Analyzer (Spark) nodes	2	The analyzer engine, which is powered by Apache Spark, analyzes this data according to defined analytic queries. This will usually follow a pattern of retrieving data from the datastore, performing a data operation such as an addition, and storing the data back in the datastore. The analyzer operations are performed by the analyzer nodes.
Dashboard nodes	1	The dashboard sends queries to the datastore for the analyzed data and displays them graphically. This function can be distributed to the dashboard nodes.



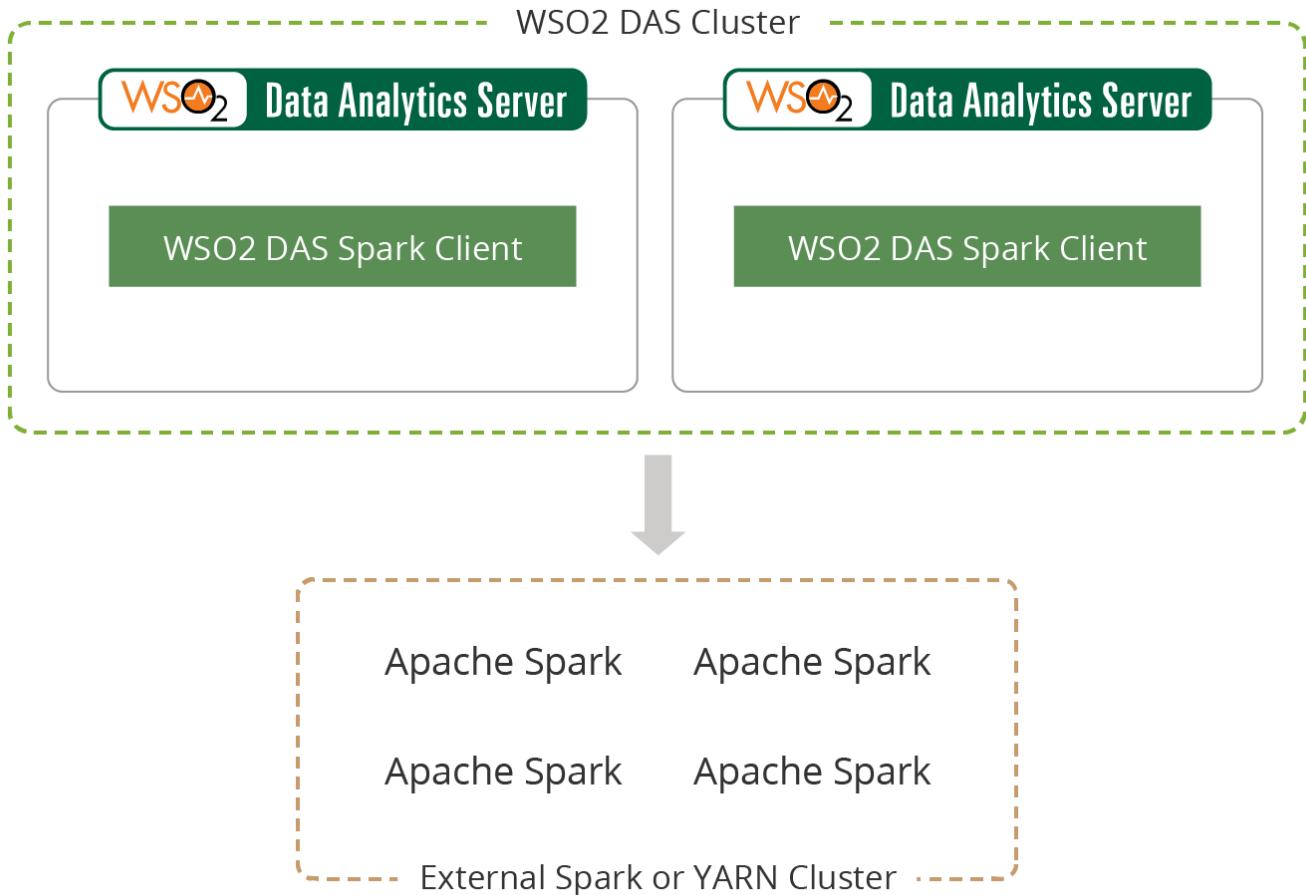
Analyzer node with external Spark or YARN cluster

WSO2 Data Analytics Server can be deployed as an Analyzer node with an external Spark or YARN cluster. This is suitable for scenarios where you want to submit WSO2 DAS analytics jobs to an external Spark cluster.



High availability analyzer cluster with external Spark or YARN cluster

WSO2 Data Analytics Server can be deployed as an Analyzer cluster for high availability with an external Spark or YARN cluster. This is suitable for scenarios where you want to submit WSO2 DAS analytics jobs to an external Spark cluster.



Spark Deployment Patterns

The embedded Spark server in WSO2 Data Analytics Server can be used in several deployment modes, depending on your requirement.

Mode	Description	When to use
Local (default)	In this mode, all of the Spark related work is done within a single node/JVM.	This is ideally suited for evaluation purposes and testing Spark queries in DAS.
Cluster (recommended)	DAS creates its own Spark cluster in the Carbon environment (using Hazelcast). This mode can be used with several high availability (HA) clustering patterns to handle failover scenarios. Additionally, in the Cluster mode, DAS can be setup without a Spark application. This allows other components to use the DAS cluster as an external Spark cluster. For example, WSO2 Machine Learner can use the Spark cluster embedded in WSO2 DAS.	For clustered production setups.

Client	In this mode, DAS acts only as a Spark client pointing to a separate Spark master.	This is suited to scenarios where you want to submit DAS analytics jobs to an external Spark cluster.
--------	------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------

The following topics list out the configuration instructions for the different deployment modes and also provide instructions on disabling Spark applications.

- Local mode
- Cluster mode
- Client mode
- Disabling a Spark application

Local mode

This is the default mode for a typical DAS instance. This mode enables users to evaluate Spark analytics in the Data Analytics Server. In this mode, a separate master or worker is not spawned. Instead, everything would run on a single JVM. Therefore, certain options like Spark Master UI and Spark Worker UI are not active.

Do the following to configure local mode.

1. **Ensure that Carbon clustering is disabled.** To do this, open the <DAS_HOME>/repository/conf/axis2/axis2.xml file and set enable="false" as shown below.

```
<clustering
    class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
    enable="false">
```
2. **Set the Spark master to local.** To do this, open the <DAS_HOME>/repository/conf/analytics/spark/spark-defaults.conf file and add the following entry (unless it already exists).

```
carbon.spark.master local[<number of cores>]
```

Cluster mode

Cluster mode is the recommended deployment pattern for DAS in the production environment. Here, DAS would create its own Spark cluster using the Carbon environment and Hazelcast. In this clustering approach, the Spark Standalone mode is used along with a custom implementation of the Standalone Recovery Mode API in Spark.

Since this mode uses a custom standalone recovery mode, the following configurations are passed into the server by default.

```
# Standalone Cluster Configs
spark.deploy.recoveryMode CUSTOM
spark.deploy.recoveryMode.factory
org.wso2.carbon.analytics.spark.core.deploy.AnalyticsRecoveryModeFactory
```

Do the following to configure cluster mode.

1. **Enable Carbon clustering.** To do this, in the <DAS_HOME>/repository/conf/axis2/axis2.xml set enable="true" for clustering as shown below.

```
<clustering
    class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
    enable="true">
```
2. In the <DAS_HOME>/repository/conf/analytics/spark/spark-defaults.conf file, set the number of masters in the cluster by adding the following entry.

```
carbon.spark.master.count <number of masters in the cluster>
```

While configuring this, make a note of the following.

- Ensure that the “carbon.spark.master local” configuration remains unchanged. This acts as a flag to use Carbon clustering.
- Each node can start as both a master and worker. So, in a two node cluster there would be two masters and two workers, one of the master nodes is active and the other is passive. You must specify the total number of masters in the cluster based on your requirement.

Client mode

Client mode is where DAS submits all the Spark related jobs to an external Spark cluster. Since this uses an external Spark cluster, you must ensure that all the .jar files required by the Carbon Spark App are included in the Spark master's and worker's SPARK_CLASSPATH.

Do the following to configure client mode.

- In the <DAS_HOME>/repository/conf/analytics/spark/spark-defaults.conf file, add the following entry.
carbon.spark.master spark://<host1:port1, host2:port2, ...>

You can include a single master or a list of masters to ensure high-availability.

Disabling a Spark application

In addition to the above modes, you can also configure DAS to startup without a Spark application. Up until the current Spark version (version 1.4.0), [there can only be one active SparkContext inside a single JVM](#). Therefore, it is not possible to allow multiple Spark applications to be created in a single JVM. Furthermore, [by default, applications submitted to the standalone mode cluster run in FIFO \(first-in-first-out\) order](#), and each application attempts to use all available nodes. The Carbon Spark application used for DAS analytics runs throughout the lifetime of the DAS cluster. Therefore, even if you create a separate Spark application in a different JVM, it can only use the resources in the cluster when the Carbon Spark application is terminated.

In order to allow other clients to use the DAS Spark cluster, there is an option provided to disable this Carbon Spark application. This can be done by setting a system variable in the server startup. See [Disabling DAS components](#) in the DAS documentation for more information.

Working with Product Specific Analytics Profiles

WSO2 implements customised Analytics functionality in WSO2 products such as WSO2 ESB, WSO2 API Manager and WSO2 Identity Server. This functionality is available by default in the WSO2 Analytics products that are used in combination with these products (e.g., the Analytics functionality specific for WSO2 ESB is available by default in WSO2 ESB Analytics). In addition, this customised functionality can be installed as features in other Analytics products (i.e., in WSO2 DAS and any other WSO2 product based on it).

Click on the relevant tab for instructions to install and use the required Analytics features in the selected Analytics product.

ESB Analytics Features	API Manager Analytics Features	Identity Server Analytics Features
----------------------------------------	------------------------------------------------	----------------------------------------------------

ESB Analytics features allow you to monitor WSO2 ESB in the Analytics Dashboard by viewing statistics for all/selected ESB artifacts. For detailed information about using ESB Analytics to monitor WSO2 ESB using ESB Analytics, see [WSO2 ESB Documentation - WSO2 ESB Analytics](#).

The ESB Analytics features can be installed and used as described below in the following WSO2 products.

- WSO2 Data Analytics Server
- WSO2 API Manager Analytics
- WSO2 IS Analytics

Installing the features

Follow the procedure below to install the required feature for ESB Analytics.

1. Log into the Management Console of your Analytics product. For detailed instructions to access the Management Console of a WSO2 product, see [Running the Product](#).
2. Add the latest P2 repository. The repository URL is given below.
<http://product-dist.wso2.com/p2/carbon/releases/wilkes/>
For detailed instructions to add a repository, see [WSO2 Admin Guide - Managing the Feature Repository](#).
3. Install the feature named `ESB Analytics`. For detailed instructions to install a feature, see [WSO2 Admin Guide - Installing Features](#).

Home > Configure > Features [Help](#)

Feature Management

Available Features

Find new features or updates to installed features in available repositories

Features	Version	Actions
<input checked="" type="checkbox"/> ESB Analytics	5.0.0	More Info.

Enabling mediation statistics

This section involves doing the required configurations to allow the WSO2 ESB server to publish statistics in the WSO2 Analytics server.

Open the `<ESB_HOME>/repository/conf/synapse.properties` file and configure the required properties as described in the table below.

Property	Required Value	Purpose
<code>mediation.flow.statistics.enable</code>	true	<p>Setting this property to true will enable the mediation statistics. This property is used to collect the following information being processed by the mediation flow:</p> <ul style="list-style-type: none"> • The time spent on each message exchange • The time spent to process the message payload before an operation is performed by individual mediation components • The fault count of a single mediation component
<code>mediation.flow.statistics.tracer.collect.payloads</code>	true	<p>Setting this property to true will enable the mediation statistics tracer to collect the message payload before an operation is performed by individual mediation components.</p>

mediation.flow.statistics.tracer.collect.properties	true	Setting this property to true enables collecting the following information being traced: <ul style="list-style-type: none">• Message context properties• Message transport-scopes
mediation.flow.statistics.collect.all	true/false	If this property is set to true, it collects statistics for all the artifacts in the system. This is enabled for all the artifacts by default. You can enable statistics for specific artifacts by setting this property to false. If this property is also set to false, then the tracing needs to be managed by marking the required artifacts as explained in the Documentation - Enabling statistics for specific artifacts .

For detailed information, see [WSO2 ESB Documentation - Prerequisites to Publish Statistics](#).

Accessing the published statistics

Follow the steps below to access the ESB Analytics dashboard.

1. Start WSO2 Analytics server, and then start the WSO2 ESB server. For detailed instructions to start a WSO2 product, see [Running the Product](#).

- WSO2 ESB is by default configured to publish statistics in a server of which the Thrift port is 7612. Therefore, you should check the default ports of your Analytics server and configure a port offset if required.

e.g., The default Thrift port of WSO2 DAS is 7611. Therefore, if you are installing ESB Analytics features in WSO2 DAS to Analyze ESB statistics, you should start the WSO2 DAS server with a port offset of 1. No port offset is needed if you are using WSO2 IS Analytics because the default Thrift port of this product is 7612.

For more information, see [WSO2 Admin Guide - Changing the Default Ports](#).

- If you run your WSO2 Analytics product on a Carbon port other than 9444 (and as a result on a Thrift port different to 7612), you should change the receiver URL specified in the following files to match the current Thrift port.
 - <ESB_HOME>/repository/deployment/server/eventpublishers/MessageFlowConfigurationPublisher.xml
 - <ESB_HOME>/repository/deployment/server/eventpublishers/MessageFlowStatisticsPublisher.xml

e.g., If you are running WSO2 Analytics on the Carbon port 9446, the Thrift port is 7614. Therefore, the receiver URL should be configured as follows in both the files mentioned above.

```
<property name="receiverURL">tcp://localhost:7614</property>
```

2. Access the ESB Analytics dashboard using the following URL.

https://<ANALYTICS_HOST>:<ANALYTICS_PORT>/portal/dashboards/esb-analytics/

For detailed information about analyzing the ESB using the gadgets in this dashboard, see [WSO2 ESB Documentation - Analyzing WSO2 ESB with the Analytics Dashboard](#).

API Manager Analytics features allow you to monitor the WSO2 API Manager as follows.

- Analyze publisher statistics using Batch Analytics. For more information, see [WSO2 API Manager Documentation - Viewing API Statistics](#).
- Configure and use Geolocation based statistics. For more information, see [WSO2 API Manager Documentation - Using Geolocation Based Statistics](#).
- Monitor APIs by receiving alerts. For more information, see [WSO2 API Manager Documentation - Managing Alerts with Real Time Analytics](#).
- Analyze Logs. For more information, see [WSO2 API Manager Documentation - Analyzing Logs with the Log Analyzer](#).

The API Manager Analytics features can be installed and used as described below in the following WSO2 products.

- WSO2 Data Analytics Server
- WSO2 ESB Analytics
- WSO2 IS Analytics

Installing the features

Follow the procedure below to install the required feature for API Manager Analytics.

1. Log into the Management Console of your Analytics product. For detailed instructions to access the Management Console of a WSO2 product, see [Running the Product](#).
2. Add the latest P2 repository. The repository URL is given below.

<http://product-dist.wso2.com/p2/carbon/releases/wilkes/>

For detailed instructions to add a repository, see [WSO2 Admin Guide - Managing the Feature Repository](#).

3. Install the following features

- Analytics Spark Geolocation UDF
- Analytics Spark scripts Common
- Analytics Spark Useragent UDF
- APIM Analytics Server Core
- APIM Analytics Server UDF

<input type="checkbox"/> Analytics Spark	1.3.0	More Info.
<input checked="" type="checkbox"/> Analytics Spark Geolocation UDF	1.0.3	More Info.
<input checked="" type="checkbox"/> Analytics Spark scripts Common	1.0.3	More Info.
<input checked="" type="checkbox"/> Analytics Spark Useragent UDF	1.0.3	More Info.
<input type="checkbox"/> Analytics Util	1.0.2	More Info.
<input type="checkbox"/> Analytics Web Service	1.3.0	More Info.
<input type="checkbox"/> Analytics Web Service Server	1.3.0	More Info.
<input checked="" type="checkbox"/> APIM Analytics Server Core	2.1.2	More Info.
<input checked="" type="checkbox"/> APIM Analytics Server UDF	2.1.2	More Info.

For detailed instructions to install APIM Analytics features, see [WSO2 API Manager Documentation - Installing WSO2 APIM Analytics Features](#).

Enabling Analytics

This section involves doing the required configurations to allow the WSO2 API Manager server to publish statistics in the WSO2 Analytics server.

1. Open the <APIM_HOME>/repository/conf/api-manager.xml file and set the Enabled property under Analytics to true as shown below. Then save this change.

```
<Enabled>true</Enabled>
```

2. If you want to enable the Log Analyzer, open the <APIM_HOME>/repository/conf/log4j.properties file and add the DAS_AGENT to the end of the root logger as shown below. Then save this change.

```
log4j.rootLogger=<other loggers>, DAS_AGENT
```

For detailed information, see [WSO2 API Manager Documentation - Configuring APIM Analytics](#).

Accessing the published statistics

Follow the steps below to access the WSO2 API Manager Portal, and the APIM Log Analyzer.

Accessing the APIM Portal

The APIM Portal allows you to configure alerts for WSO2 API Manager so that you can be notified when different statistics relating to your APIs reach specified limits. As a result, you can effectively manage your APIs.

1. Start WSO2 DAS server, and then start the WSO2 APIM server. For detailed instructions to start a WSO2 product, see [Running the Product](#).

- WSO2 API Manager is by default configured to publish statistics in a server of which the Thrift port is 7612. Therefore, you should check the default ports of your Analytics server and

configure a port offset if required.

e.g., The default Thrift port of WSO2 DAS is 7611. Therefore, if you are installing API Manager Analytics features in WSO2 DAS to Analyze APIM statistics, you should start the WSO2 DAS server with a port offset of 1. No port offset is needed if you are using WSO2 IS Analytics because the default Thrift port of this product is 7612.

For more information, see [WSO2 Admin Guide - Changing the Default Ports](#).

- If you run your WSO2 Analytics product on a Carbon port other than 9444 (and as a result on a Thrift port different to 7612), you should change the receiver URL specified in the <APIM_HOME>/repository/conf/api-manager.xml file to match the current Thrift port.

e.g., If you are running WSO2 Analytics on the Carbon port 9446, the Thrift port is 7614. Therefore, the receiver URL should be configured as follows in the <APIM_HOME>/repository/conf/api-manager.xml file.

```
<DASServerURL>{tcp://localhost:7614}</DASServerURL>
```

2. Log into the WSO2 API Manager Admin Portal using the https://localhost:<SERVER_PORT>/adminUI.
3. Click **Settings** to expand that section and then click **Analytics**. This opens the **Alert Configurations** page where all the supported alert types are displayed. For details information to configure different alerts, see [WSO2 API Manager Documentation- Configuring Alerts](#).

Accessing the Log Analyzer

The Log Analyzer allows you analyze messages logged for WSO2 APIM.

1. Start WSO2 DAS server, and then start the WSO2 APIM server. For detailed instructions to start a WSO2 product, see [Running the Product](#).
2. Access the WSO2 API Manager Admin Portal using the https://<Host_Name>:<Port_Name>/adminUI. Log into the portal using your credentials.
3. In the left navigator, click **Log Analyzer** to expand the **Log Analyzer** section. Then click on the relevant pages to perform your analysis as required. For detailed information about each page, see [WSO2 API Manager Documentation - Analyzing Logs with the Log Analyzer](#).

Identity Server Analytics features allow you to monitor the authentication operations performed by WSO2 IS using the Analytics Dashboard. For more information, See [WSO2 IS Documentation - Analyzing Statistics for Authentication Operations](#).

The IS Analytics features can be installed and used as described below in the following WSO2 products.

- WSO2 Data Analytics Server
- WSO2 ESB Analytics
- WSO2 API Manager Analytics

Installing the features

Follow the procedure below to install the required feature for IS Analytics.

1. Log into the Management Console of your Analytics product. For detailed instructions to access the Management Console of a WSO2 product, see [Running the Product](#).
2. Add the latest P2 repository. The repository URL is given below.
<http://product-dist.wso2.com/p2/carbon/releases/wilkes/>
For detailed instructions to add a repository, see [WSO2 Admin Guide - Managing the Feature Repository](#).
3. Install the feature named IS Analytics. For detailed instructions to install a feature, see [WSO2 Admin Guide - Installing Features](#).

Home > Configure > Features [Help](#)

Feature Management

Available Features Installed Features Installation History Repository Management

Available Features

Find new features or updates to installed features in available repositories

Repository:		P2 – http://product-dist.wso2.com/p2/carbon/releases/wilkes/	Add Repository
Filter by feature name:		<input type="text" value="IS Analytics"/>	
<input checked="" type="checkbox"/> Show only the latest versions		<input type="checkbox"/> Group features by category	
Find Features			

Select all in this page | Select none [Install](#)

Features	Version	Actions
<input checked="" type="checkbox"/> IS Analytics	5.2.0	More Info.

Select all in this page | Select none [Install](#)

Enabling Analytics

This section involves doing the required configurations to allow the WSO2 IS server to publish statistics in the WSO2 Analytics server.

1. Open the `IS_HOME>/repository/conf/identity/Identity.xml` file.
2. Enable the `org.wso2.carbon.identity.data.publisher.application.authentication.AuthenticationDataPublisherProxy` listener as follows. This listener needs to be enabled in order to publish statistics relating to both logins and sessions.

```
<EventListener
type="org.wso2.carbon.identity.core.handler.AbstractIdentityMessageHandler"
name="org.wso2.carbon.identity.data.publisher.application.authentication.AuthenticationDataPublisherProxy"
orderId="11" enable="true" />
```

3. Enable the `org.wso2.carbon.identity.data.publisher.application.authentication.impl.DASLoginDataPublisherImpl` listener as follows if you want statistics relating to logins published in the Analytics Dashboard.

```
<EventListener
type="org.wso2.carbon.identity.core.handler.AbstractIdentityMessageHandler"
name="org.wso2.carbon.identity.data.publisher.application.authentication.impl.DAS
LoginDataPublisherImpl"
orderId="10" enable="true" />
```

4. Enable the `org.wso2.carbon.identity.data.publisher.application.authentication.impl.DASSessionDataPublisherImpl` listener as follows if you want statistics relating to sessions to be published in the Analytics Dashboard.

```
<EventListener
type="org.wso2.carbon.identity.core.handler.AbstractIdentityMessageHandler"
name="org.wso2.carbon.identity.data.publisher.application.authentication.impl.DAS
SessionDataPublisherImpl"
orderId="11" enable="true" />
```

For detailed information, see [WSO2 IS Documentation - Prerequisites to Publish Statistics](#).

Accessing the published statistics

Follow the steps below to access the IS Analytics dashboard.

1. Start WSO2 Analytics server, and then start the WSO2 IS server. For detailed instructions to start a WSO2 product, see [Running the Product](#).

- WSO2 IS is by default configured to publish statistics in a server of which the Thrift port is 7612. Therefore, you should check the default ports of your Analytics server and configure a port offset if required.

e.g., The default Thrift port of WSO2 DAS is 7611. Therefore, if you are installing IS Analytics features in WSO2 DAS to Analyze IS statistics, you should start the WSO2 DAS server with a port offset of 1. No port offset is needed if you are using WSO2 ESB Analytics because the default Thrift port of this product is 7612.

For more information, see [WSO2 Admin Guide - Changing the Default Ports](#).

- If you run your WSO2 Analytics product on a Carbon port other than 9444 (and as a result on a Thrift port different to 7612), you should change the receiver URL specified in the following files to match the current Thrift port.

- `<IS_HOME>/repository/deployment/server/eventpublishers/IsAnalytics-Publisher-wso2event-AuthenticationData.xml`
- `<IS_HOME>/repository/deployment/server/eventpublishers/IsAnalytics-Publisher-wso2event-SessionData.xml`

e.g., If you are running WSO2 Analytics on the Carbon port 9446, the Thrift port is 7614. Therefore, the receiver URL should be configured as follows in both the files mentioned above.

```
<property name="receiverURL">tcp://localhost:7614</property>
```

2. Access the IS Analytics dashboard using the following URL.

https://<ANALYTICS_HOST>:<ANALYTICS_PORT>/portal/dashboards/is-analytics/

For detailed information about analyzing the IS using the gadgets in this dashboard, see [WSO2 IS Documentation - Analyzing Statistics for Authentication Operations](#).

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>/libs/ 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.

```
org.osgi.framework.system.packages=javax.accessibility,\njava.awt.awt,\n...\njava.imageio.stream,\njavax.management,\n...\n
```

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 DAS.

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>
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.
 - a. orbit
 - b. siddhi
 - c. 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 DAS.

1. Download WSO2 DAS, and start the server. For instructions, see [Getting Started](#).
2. Log into the DAS 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.

You can add a new local repository or a remote repository

Name: *

Location:

- URL e.g. <http://dist.wso2.org/p2/carbon/releases/4.4.1-SNAPSHOT>
- Local e.g. <C:/userrepo>, </home/user/p2-repo>

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.

Select all in this page | Select none

Features	Version	Actions
<input type="checkbox"/> GPL - Siddhi Eval Script Extension	2.0.4.SNAPSHOT	More Info.
<input type="checkbox"/> GPL - Siddhi Geo Extension	2.0.4.SNAPSHOT	More Info.
<input type="checkbox"/> GPL - Siddhi NLP Extension	2.0.4.SNAPSHOT	More Info.
<input type="checkbox"/> GPL - Siddhi PMML Extension	2.0.4.SNAPSHOT	More Info.
<input type="checkbox"/> GPL - Siddhi R Extension	2.0.4.SNAPSHOT	More Info.

Select all in this page | Select none

10. Once the features are selected, click **Install** to proceed with the installation.
11. Once the installation is completed, [restart WSO2 DAS](#).

Changing the Host Name

The contents of this page are currently in progress.

The following procedure explains how to change the host name of DAS as required for your production environment. In this example, the hostname needs to be changed to node1.analytics.com.

1. Open the <DAS_HOME>/repository/conf/carbon.xml file and set the HostName property as shown below.

```
<HostName>node1.analytics.com</HostName>
```

2. Open DAS_Home/repository/deployment/server/jaggeryapps/portal/configs/designer.json and change the hostname to the actual hostname of WSO2 IS Analytics.

```
},
"host": {
    "hostname": "localhost",
    "port": "",
    "protocol": ""
}
```

This step avoids the host verification errors that can occur when another WSO2 product publishes statistics in WSO2 DAS.

3. Generate a key store by following the sub steps given below.

- a. Open a terminal and issue the following command to generate a key store.

```
keytool -genkey -alias node1.analytics.com -keyalg RSA -keystore
analyticsnode1.jks -keysize 2048
```

- b. Specify a preferred KeyStore password once prompted.

- c. Specify a preferred Key password once prompted.

- d. Enter the first name and last name as *.node1.analytics.com once prompted.

- e. Enter values for the other parameters as required.

Once this information is submitted, a key store is generated with a private key and a public certificate with no de1.analytics.com as the CN.

4. Copy the generated self-signed key store (i.e., analyticsnode1.jks) to the <DAS_HOME>/repository/resources/security directory.

5. Export the public certificate from the keystore and import that certificate to the client--truststore.jksfile following the steps given below.

- a. Navigate to the <DAS_HOME>/repository/resources/security directory.

- b. Issue the following command to export the public certificate from the primary key store

```
keytool -export -alias node1.analytics.com -file node1.analytics.com -keyst
ore analyticsnode1.jks -storepass <keystore_password_given_above>
```

- c. Issue the following command to import the certificate to the client--truststore.jksfile.

```
keytool -import -alias node1.analytics.com -file
node1.analytics.com -keystore client-truststore.jks -storepass wso2carbon
```

Product Administration

WSO2 Data Analytics Server (WSO2 DAS) is shipped with default configurations that allow you to download, install and get started with your product instantly. However, when you go into production, it is recommended to change some of the default settings to ensure that you have a robust system that is suitable for your operational needs. Also, you may have specific use cases that require specific configurations to the server. If you are a product administrator, the follow content provides you with an overview of the administration tasks that you need to perform when working with Data Analytics Server (WSO2 DAS).

[[Upgrading from a previous release](#)] [[Changing the default database](#)] [[Configuring users, roles and permissions](#)] [[Configuring security](#)] [[Configuring transports](#)] [[Configuring multitenancy](#)] [[Performance tuning](#)] [[Changing the default ports](#)] [[Managing product features](#)] [[Customizing the Management Console](#)] [[Applying Patches](#)] [[Monitoring the server](#)]

[Upgrading from a previous release](#)

If you are upgrading from WSO2 DAS 3.0.1 to WSO2 DAS 3.1.0, see [Upgrading from WSO2 DAS 3.0.1](#).

[Changing the default database](#)

To change the default database configurations for WSO2 DAS, see [Working with Databases](#).

[Configuring users, roles and permissions](#)

The user management feature in your WSO2 DAS allows you to create new users and define the permissions granted to each user. You can also configure the user stores that are used for storing data related to user management.

- For instructions on how to configure user management, see [WSO2 Administration Guide - Working with Users, Roles and Permissions](#).
- For descriptions of permissions that apply to WSO2 DAS users, see [WSO2 DAS - Specific User Permissions](#).

[Configuring security](#)

After you install WSO2 DAS, it is recommended to change the default security settings according to the requirements of your production environment. As WSO2 DAS is built on top of the WSO2 Carbon Kernel (version 4.4.x), the main security configurations applicable to WSO2 DAS are inherited from the Carbon kernel.

For instructions on configuring security in your server, see the following topics in the WSO2 Administration Guide.

- [Configuring Transport-Level Security](#)
 - [Using Asymmetric Encryption](#)
 - [Using Symmetric Encryption](#)
 - [Enabling Java Security Manager](#)
 - [Securing Passwords in Configuration Files](#)
 - [Resolving Hostname Verification](#)
-

[Configuring transports](#)

The WSO2 Carbon Kernal on which WSO2 DAS is built on supports a variety of transports.

For instructions on configuring these transports, see [WSO2 Administration Guide - Working with Transports](#).

[Configuring multitenancy](#)

You can create multiple tenants in your DAS server which allows you to maintain tenant isolation in a single server/cluster. For instructions on configuring multiple tenants for your server, see [WSO2 Administration Guide - Working with Multiple Tenants](#).

Configuring the registry

A **registry** is a content store and a metadata repository for various artifacts such as services, WSDLs and

configuration files. In WSO2 products, all configurations pertaining to modules, logging, security, data sources and other service groups are stored in the registry by default.

For instructions on setting up and configuring the registry for your server, see [WSO2 Administration Guide - Working with the Registry](#).

Performance tuning

You can optimize the performance of your WSO2 server by using configurations and settings that are suitable to your production environment. At a basic level, you need to have the appropriate OS settings, JVM settings etc. Since WSO2 products are all based on a common platform called Carbon, most of the OS, JVM settings recommended for production are common to all WSO2 products. Additionally, there are other performance enhancing configuration recommendations that depend on very specific features used by your product.

For instructions on the Carbon platform-level performance tuning recommendations, see [WSO2 Administration Guide - Performance Tuning](#).

For instructions to tune performance for a Spark cluster, see [Tuning Performance for a Spark Cluster](#).

For instructions to tune performance for real time analytics, see [Tuning Performance for Real Time Analytics](#).

Changing the default ports

When you run multiple WSO2 products, multiple instances of the same product, or multiple WSO2 product clusters on the same server or virtual machines (VMs), you must change their default ports with an offset value to avoid port conflicts.

For instructions on configuring ports, see [WSO2 Administration Guide - Changing the Default Ports](#).

Managing product features

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. By default, WSO2 DAS is shipped with the features that are required for your main use cases.

For information on installing new features, or removing/updating an existing feature, see [WSO2 Administration Guide - Working with Features](#).

Configuring custom proxy paths

This feature is particularly useful when multiple WSO2 products (fronted by a proxy server) are hosted under the same domain name. By adding a custom proxy path you can host all products under a single domain and assign proxy paths for each product separately .

For instructions on configuring custom proxy paths, see [WSO2 Administration Guide - Adding a Custom Proxy Path](#).

Customizing error pages

You can make sure that sensitive information about the server is not revealed in error messages, by customizing the error pages in your product.

For instructions, see [WSO2 Administration Guide - Customizing Error Pages](#).

Customizing the Management Console

Some of the WSO2 products, such as WSO2 DAS consist of a web user interface named the management console. This allows administrators to configure, monitor, tune, and maintain the product using a simple interface. You can customize the look and feel of the management console for your product.

For instructions, see [WSO2 Administration Guide - Customizing the Management Console](#).

Applying Patches

For instructions on applying patches (issued by WSO2), see [WSO2 Administration Guide - WSO2 Patch Application Process](#).

Monitoring the server

Monitoring is an important part of maintaining a product server. Monitoring capabilities available for WSO2 DAS are listed below.

- **Monitoring server logs:** A properly configured logging system is vital for identifying errors, security threats and usage patterns in your product server. For instructions on monitoring the server logs, see [WSO2 Administration Guide - Monitoring Logs](#).
- **Monitoring Spark logs:** WSO2 DAS generates logs to monitor the performance of Spark workers in a clustered set up. For more information, see [Monitoring Spark Performance](#).
- **Monitoring using JVM Metrics:** WSO2 DAS is shipped with Metrics that allows you to monitor statistics of your server using Java Metrics. For instructions on setting up and using Carbon metrics for monitoring, see [WSO2 Administration Guide - Using WSO2 Carbon Metrics](#).
- **JMX-based monitoring:** For information on monitoring your server using JMX, see [WSO2 Administration Guide - JMX-based monitoring](#).

Samples

This section demonstrates a few practical examples of the WSO2 Data Analytics Server, their objectives and expected behavior.

- [Real Time Samples](#)
- [Sending Notifications Through Published Events Using Spark](#)
- [Analyzing HTTPD Logs](#)
- [Analyzing Smart Home Data](#)
- [Analyzing Realtime Service Statistics](#)
- [Analyzing Wikipedia Data](#)

Real Time Samples

The following topic cover the real time samples available for WSO2 DAS.

- [Setting Up Real Time Samples](#)
- [WSO2 DAS Real Time Samples](#)

Setting Up Real Time Samples

For information on understanding the general flow of DAS real time samples, see [WSO2 CEP Samples](#). The following sections explain the generic setup instructions to execute the samples.

- Prerequisites
- Starting sample 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 DAS real time samples in this documentation.

Sample	Requirements
All samples	<ul style="list-style-type: none"> • 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 Data Analytics Server
JMS related samples	<p>The JMS samples are explained to be tried out using following JMS providers</p> <ul style="list-style-type: none"> • 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	<p>The MQTT samples are explained to be tried out using following MQTT-supported servers</p> <ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> • 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 configurations

To start the WSO2 DAS with a sample configuration, run the following command with `-sn <n>`, where `<n>` denotes the number assigned to the sample.

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

For example, to start the WSO2 DAS with the configuration of sample 0101, run the following command inside `<DAS_HOME>/bin` directory:

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

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

In the normal mode the `<DAS_HOME>/bin/wso2server.bat` or `<DAS_HOME>/bin/wso2server.sh` script starts an instance of the DAS using the configuration files in `<DAS_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 `<DAS_HOME>/samples/cep/artifacts/<SAMPLE_NUMBER>/` directory. (`<DAS_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 `<DAS_HOME>/samples/cep/consumers/<consumer_name>`.

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

```

user@host:/tmp/wso2das-3.1.0/samples/consumers/wso2-event$ ant
Buildfile: /home/user/tmp/wso2das-3.1.0/samples/consumers/wso2-event/build.xml
...
run:[echo] To configure host, port and events use -Dhost=xxxxx -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:

```
ant -DsmapleNo=<sample no>
```

Running `DsampleNo` 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 DAS server.

```

user@host:/tmp/wso2das-3.1.0/samples/consumers/logService$ ant -DsmapleNo=0102
Buildfile: /home/usre/tmp/wso2das-3.1.0/samples/consumers/logService/build.xml
-folder.check:
-assign.sample:
[echo] Sample No : 0102
[echo] Services Dir : ../../samples/artifacts/0102/axis2services
-assign.main:
folder.set:
clean:
...
[jar] Building jar:
/tmp/wso2das-3.1.0/repository/deployment/server/webapps/logService.war
BUILD SUCCESSFUL
Total time: 0 seconds

```

Starting sample producers

Starting a sample producer is similar to starting a consumer.

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

```

user@host:/home/user/wso2das-3.1.0/samples/producers/pizza-shop$ ant
pizzaOrderClient -Dservice=WSEventLocalAdaptorService
-DtopicName=BatchedPizzaOrder -DbatchedEvents=true
Buildfile: /home/user/tmp/wso2das-3.1.0/samples/producers/pizza-shop/build.xml
init:
compile:
[copy] Copying 1 file to
/home/user/tmp/wso2das-3.1.0/samples/producers/pizza-shop/temp/classes
pizzaOrderClient:
[echo] To configure host and port use -Dhost=xxxx -Dport=xxx -Dservice=xxx
-DtopicName=xxx
[echo] Sending to :
http://localhost:9763/services/WSEventLocalAdaptorService/BatchedPizzaOrder

[echo] To send events in batches use -DbatchedEvents=true
[echo] Sending events in batches : true
BUILD SUCCESSFUL
Total time: 1 second

```

Passing arguments to sample clients

Some sample clients take extra arguments. The following table presents the format in which these arguments can be passed.

Purpose	Syntax	Example
To specify the publishing topic for the producer client.	-DtopicName=XXXX	ant -DtopicName=AllStockQuotes
To publish to a specific host, which is an IP address.	-Dhost=XXXX	ant -Dhost=org.test.domain
To publish to a specific port.	-Dport=XXXX	ant pizzaOrderClient -Dport=9764
To publish to a specific Web service.	-Dservice=XXXX	ant pizzaOrderClient -Dservice=wsInAd.
To send events in batches (i.e., the adapter receives a batch of events).	-DbatchedEvents={true false}	ant -DbatchedEvents=true
To publish events to a specific client URL.	-Durl='client url'	-Durl=http://localhost:9763/endpoints

To subscribe events from a JMS topic (consumer).	-DtopicName=XXXXXX	ant topicConsumer -DtopicName=TestTop
To subscribe events from a JMS queue (consumer).	-Dqueue=XXXXX	ant queueConsumer -Dqueue=DelayedFlight
To receive events from a specific format from the text document (producer).	-Dformat=xxxx(csv, text, json, xml)	ant -Dformat=csv
To specify the JMS broker to which the DAS server listens.	-Dbroker=xxxx(activemq, qpid)	ant -Dbroker=activemq
To publish events in a specific event stream (producer).	-DstreamId=xxxx:x.x.x	ant -DstreamId=org.wso2.event.sensor.
To publish events from the specific sample folder (producer).	-Dsn='sample number' or -DfilePath=xxxx	ant -Dsn=00
To specify whether the protocol based on which events are received is thrift or binary.	-Dprotocol='thrift/binary'	ant -Dprotocol=binary
To specify the username when an action performed by a sample requires user credentials to be specified.	-Dusername=xxxx	-Dusername=admin
To specify the password when an action performed by a sample requires user credentials to be specified.	-Dpassword=xxxx	-Dpassword=admin

When doing a performance test, this argument specifies the number of events with which the test should be carried out.	-Devents=xx or -DnoOfEvents=xxxx or -DeventCount=xxxx	-Devents=2000000
When doing a performance test, this argument specifies the delay that occurs between events in milli seconds.	-Ddelay='delay between events in ms'	-Ddelay=1000
When doing a performance test, this argument specifies the number of events after which the throughput/latency should be calculated.	-DelapsedCount=xxxx	-DelapsedCount=10000
When doing a performance test, this argument specifies the number of publishers that should be used to publish events.	-DnoOfPublishers=xxxx	-DnoOfPublishers=50
When doing a performance test, this argument specifies the number of events that should be sent to the event flow for the DAS server to warm up and reach a stabilize.	-DwarmUpCount=xxxx	-DwarmUpCount=200000

<p>When doing a performance test, this argument is used to specify whether you want to calculate the throughput or the latency.</p> <ul style="list-style-type: none"> Throughput: This is the number of events processed concurrently at a given time by an event flow. Latency: This is the time taken by the event flow to process a single event. 	-DcalcType='throughput/latency'	-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 <DAS_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 the <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/qpidd-client-0.32.jar
- <QPID-CLIENT_HOME>/lib/qpidd-common-0.32.jar

For WSO2 Message Broker (MB) , the relevant JAR files are,

- <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
- <MB_HOME>/client-lib/org.wso2.securevault-1.0.0-wso2v2.jar

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 <DAS_HOME>/samples/cep/lib folder as mentioned below.

- Download and add [MQTT client library \(mqtt-client-0.4.0.jar\)](#) to <DAS_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 <DAS_HOME>/samples/cep/lib folder as mentioned below.

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

WSO2 DAS Real Time Samples

WSO2 DAS 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 DAS with a different configuration.
2. After WSO2 DAS starts, the sample producers send different types of events to the DAS over different transports.
3. The DAS 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 real time samples, demonstrating use cases of WSO2 DAS.

- [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 DAS sample. For instructions to set up the samples, see [Setting Up Real Time Samples](#).

Samples on receiving events

Sample no.	Description	Producer (publish topic)	Event receiver type	Receiver message format	Event publisher type	Publisher message format	Consumer (subscription topic)
0001	Receiving JSON events via HTTP transport.	http	HTTP	JSON	logger	text	None
0002	Receiving custom JSON events via HTTP transport.	http	HTTP	JSON (custom)	logger	text	None
0003	Receiving XML events via HTTP transport.	http	HTTP	XML	logger	text	None
0004	Receiving custom XML events via HTTP transport.	http	HTTP	XML (custom)	logger	text	None
0005	Receiving text events via HTTP transport.	http	HTTP	text	logger	text	None

0006	Receiving custom text events via HTTP transport.	http	HTTP	text (custom)	logger	text	None
0007	Receiving WSO2 events via WSO2 Event Receiver.	wso2event	WSO2Event	WSO2Event	logger	text	None
0008	Receiving custom WSO2 events via WSO2 Event Receiver.	wso2event	WSO2Event	WSO2Event (custom)	logger	text	None
0009	Receiving map events via JMS transport - ActiveMQ.	jms	JMS (Active MQ)	map	logger	text	None
0010	Receiving custom map events via JMS transport - ActiveMQ.	jms	JMS (Active MQ)	map (custom)	logger	text	None
0011	Receiving JSON, text, XML events via JMS transport - ActiveMQ.	jms	JMS (Active MQ)	JSON/text/XML	logger	text	None
0012	Receiving map, XM events via JMS transport - Qpid	jms	JMS (Qpid)	map/XML	logger	text	None
0013	Receiving map, text events via JMS transport - WSO2 Message Broker.	jms	JMS (WSO2 MB)	map/text	logger	text	None
0014	Receiving XML events via SOAP transport.	soap	SOAP	XML	logger	text	None
0015	Receiving text events via Email transport.	None	Email	text	logger	text	None
0016	Receiving JSON events via MQTT transport.	mqtt	MQTT	JSON	logger	text	None
0017	Receiving custom text events via file tail transport.	file	file-tail	text (custom)	logger	text	None

0018	Receiving JSON events via Kafka transport.	kafka	kafka	JSON	logger	text	None
0019	Receiving JSON events via WebSocket transport.	websocket	websocket	JSON	logger	text	None
0020	Simple JSON pass-through with WebSocket local input event adaptor.	None	WebSocket local	JSON	logger	text	None
0021	Receiving map events via JMS transport - ActiveMQ (For Queue)	jms	JMS (ActiveMQ)	map	logger	text	None
0022	Receiving custom RegEx events via text events via file tail transport.	file	file-tail	text (custom)	logger	text	None

Samples on processing events

Sample no.	Description	Producer (publish topic)	Event receiver type	Receiver message format	Event publisher type	Publisher message format	Cor (sub topic)
0101	Pass-through/projection query in an execution plan.	wso2-event	WSO2Event	WSO2Event	logger	text	Non
0102	Projections, transformations and enrichment for events.	None	(Event Simulator)	None	logger	text	Non
0103	Using filters for generating alerts.	None	(Event Simulator)	None	logger	text	Non
0104	Calculations over time using windows.	None	(Event Simulator)	None	logger	text	Non
0105	Performing joins with windows.	None	(Event Simulator)	None	logger	text	Non
0106	Using in-memory event tables.	None	(Event Simulator)	None	logger	text	Non
0107	Using RDBMS event tables.	None	(Event Simulator)	None	logger	text	Non
0108	Using patterns to detect ATM transaction frauds.	None	(Event Simulator)	None	logger	text	Non

0109	Detecting trends with sequences.	None	(Event Simulator)	None	logger	text	Non
0110	Sequences with partitioning to detect trends.	None	(Event Simulator)	None	logger	text	Non
0111	Detecting non-occurrences with patterns.	http	HTTP	text	logger	text	Non
0112	Analyzing Twitter feeds using partitions.	wso2-event	WSO2Event	WSO2Event	logger	JSON	Non
0113	Limiting the output rate of an event stream.	http	HTTP	XML	logger	text	Non
0114	Using external time windows.	wso2-event	WSO2Event	WSO2Event	logger	text	Non
0115	Quartz scheduler based alerts.	http	HTTP	XML	logger	JSON	Non
0116	Performing linear regression.	None	(Event Simulator)	None	logger	text	Non
0117	Filtering and outputting to multiple streams.	file	file-tail	text	logger	text	Non
0118	Using Hazelcast event tables.	None	(Event Simulator)	None	logger	text	Non
0119	Trigger events at defined time intervals.	None	(Event Simulator)	None	logger	text	Non
0301	Network intruder detection with PMML extension predictions.	None	(Event Simulator)	None	logger	text	Non
0501	Processing a simple filter query with Apache Storm Deployment.	wso2-event	WSO2Event	WSO2Event	WSO2Event	WSO2Event	wso
0502	Processing a window state over server restart.	http	HTTP	XML	logger	text	Non
0503	Processing a window query in high availability mode.	http	HTTP	XML	logger	text	Non
0504	Processing a distributed Siddhi query with partitioning by integrating with Apache Storm.	wso2-event	WSO2Event	WSO2Event	logger	text	Non
1001	WSO2 DAS Geo Dashboard.	http	HTTP	JSON	websocket-local	JSON	Non

Samples on publishing events

Sample no.	Description	Producer (publish topic)	Event receiver type	Receiver message format	Event publisher type	Publisher message format	Consumer (subscription topic)
0051	Publishing JSON events via logger transport.	None	None	None	logger	JSON	None
0052	Publishing custom JSON events via logger transport.	None	None	None	logger	JSON (custom)	None
0053	Publishing XML events via logger transport.	None	None	None	logger	XML	None
0054	Publishing custom XML events via logger transport.	None	None	None	logger	XML (custom)	None
0055	Publishing text events via logger transport.	None	None	None	logger	text	None
0056	Publishing custom text events via logger transport.	None	None	None	logger	text (custom)	None
0057	Publishing WSO2 events via WSO2Event transport.	None	None	None	WSO2Event	WSO2Event	wso2-event
0058	Publishing custom WSO2 events via WSO2Event transport.	None	None	None	WSO2Event	WSO2Event (custom)	wso2-event
0059	Publishing map and text events via JMS transport - ActiveMQ.	None	None	None	JMS (ActiveMQ)	map/text	jms

0060	Publishing custom map and JSON events via JMS transport - Qpid.	None	None	None	JMS (Qpid)	map (custom)/JSON (custom)	jms
0061	Publishing map and XML events via JMS transport - WSO2 MB.	None	None	None	JMS (WSO2 MB)	map/XML	jms
0062	Publishing XML, JSON and custom text events via HTTP transport.	None	None	None	HTTP	JSON/text (custom)/XML	generic-log-servi
0063	Publishing XML events via SOAP transport.	None	None	None	SOAP	XML	axis2-log-service
0064	Publishing text events via Email transport.	None	None	None	email	text	
0065	Publishing JSON events via SMS transport.	None	None	None	SMS	JSON	SMSC Simulator
0066	Publishing JSON events via MQTT transport.	None	None	None	MQTT	JSON	mqtt
0067	Publishing map events via Cassandra transport.	None	None	None	Cassandra	map	
0068	Publishing XML events via Kafka transport.	None	None	None	Kafka	XML	

0069	Publishing JSON events via WebSocket transport.	None	None	None	Websocket	JSON	websocket
0070	Publishing JSON events via WebSocket local output event adapter.	None	None	None	WebSocket local	JSON	
0071	Publishing WSO2Event events via UI transport.	None	None	None	UI	WSO2Event	
0072	Publishing map events via RDBMS transport.	login-info	WSO2Event	WSO2Event	RDBMS	map	

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 or 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 adhere to the WSO2Event format via the HTTP transport. This sample does not process 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 DAS 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 name `httpLogger` to log the received messages.

Executing the sample

Open a new tab in the CLI and execute the following `ant` command from the `<DAS_HOME>/samples/cep/producers/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 `<DAS_HOME>/samples/cep/producers/http/build.xml` file.

This builds the HTTP client and publishes the events in the `<DAS_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.

```
[echo] Configure -Durl=xxxx and (-DfilePath=xxxx or -Dsn='sample number') optionally use -Dusername=xxxx -Dpassword=xxxx
[java] Starting WSO2 Http Client
[java] Sending message:
[java]
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 701,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
[java] Sending message:
[java]
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 702,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
[java] Sending message:
[java]
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 703,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
```

The logs of the JSON events received by the DAS server will be displayed in the CLI as shown in the example below.

```

Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:701,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
[2016-07-22 14:09:32,834] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:702,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
[2016-07-22 14:09:32,837] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:703,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343

```

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 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 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 <DAS_HOME>/samples/cep/producers/http directory.

```
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0002
```

It builds the http client and publishes the events in the <DAS_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] Configure -Durl=xxxx and ( -DfilePath=xxxx or -Dsn='sample number') optionally use -Dusername=xxxx -Dpassword=xxxx
[java] [main] INFO org.wso2.carbon.sample.http.Http - Starting WSO2 Http Client
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] {
[java]   "sensorData": {
[java]
[java]     "timestamp": 19900813115534,
[java]     "powerSaved": false,
[java]     "id": 501,
[java]     "name": temperature,
[java]     "long": 90.34344,
[java]     "lat": 20.44345,
[java]     "humidity": 2.3,
[java]     "temp": 20.44345
[java]   }
[java] }
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] {
[java]   "sensorData": {
[java]
[java]     "timestamp": 19900813115534,
[java]     "powerSaved": false,
[java]     "id": 502,
[java]     "name": temperature,
[java]     "long": 90.34344,
[java]     "lat": 20.44345,
[java]     "humidity": 2.3,
[java]     "temp": 20.44345
[java]   }
[java] }
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] {
[java]   "sensorData": {
[java]
[java]     "timestamp": 19900813115534,
[java]     "powerSaved": false,
[java]     "id": 503,
[java]     "name": temperature,
[java]     "long": 90.34344,
[java]     "lat": 20.44345,
[java]     "humidity": 2.3,
[java]     "temp": 20.44345
[java]   }
[java] }
[java]
```

Events received by WSO2 DAS are logged as follows.

```
[2016-03-09 11:22:53,444] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:false, meta_sensorId:501, meta_sensorName:temperature, correlation_longitude:90.34344, correlation_latitude:20.44345, humidity:2.3, sensorValue:20.44345 [2016-03-09 11:22:53,445] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:false, meta_sensorId:502, meta_sensorName:temperature, correlation_longitude:90.34344, correlation_latitude:20.44345, humidity:2.3, sensorValue:20.44345 [2016-03-09 11:22:53,451] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:false, meta_sensorId:503, meta_sensorName:temperature, correlation_longitude:90.34344, correlation_latitude:20.44345, humidity:2.3, sensorValue:20.44345
```

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 WSO2Event format via http the transport. This sample does not process 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 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 <DAS_HOME>/samples/cep/producers/http directory.

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

It builds the http client and publishes the events in the <DAS_HOME>/samples/cep/artifacts/0003/httpReceiver.txt 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
[java] [main] INFO org.wso2.carbon.sample.http.Http - Starting WSO2 Http Client
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] <events>
[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>20.44345</sensorValue>
[java]     </payloadData>
[java]   </event>
[java] </events>
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] <events>
[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>20.44345</sensorValue>
[java]     </payloadData>
[java]   </event>
[java] </events>
```

The events received by WSO2 DAS are logged as follows.

```
t[2016-03-09 12:10:16,388] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:199008131245, meta_isPowerSaverEnabled:true, meta_sensorId:4, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:1.23434, humidity:6.6, sensorValue:20.44345
[2016-03-09 12:10:16,406] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:199008131245, meta_isPowerSaverEnabled:true, meta_sensorId:4, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:1.23434, humidity:6.6, sensorValue:20.44345
```

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 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 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 <DAS_HOME>/samples/cep/producers/http directory.

```
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0004
```

This builds the HTTP client and publishes the events in the <DAS_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.

```
[echo] Configure -Durl=xxxx and ( -DfilePath=xxxx or -Dsn='sample number') optionally use -Dusername=xxxx -Dpassword=xxxx
[java] [main] INFO org.wso2.carbon.sample.http.Http - Starting WSO2 Http Client
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] <temperature:SensorStream xmlns:temperature="http://samples.wso2.org">
[java]   <temperature:sensor>
[java]     <temperature:timestamp>19900813115534</temperature:timestamp>
[java]     <temperature:powerSaved>true</temperature:powerSaved>
[java]     <temperature:id>502</temperature:id>
[java]     <temperature:name>temperature</temperature:name>
[java]     <temperature:long>4.504343</temperature:long>
[java]     <temperature:lat>1.23434</temperature:lat>
[java]     <temperature:humidity>6.6</temperature:humidity>
[java]     <temperature:temp>20.44345</temperature:temp>
[java]   </temperature:sensor>
[java] </temperature:SensorStream>
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] <temperature:SensorStream xmlns:temperature="http://samples.wso2.org">
[java]   <temperature:sensor>
[java]     <temperature:timestamp>19900813115534</temperature:timestamp>
[java]     <temperature:powerSaved>true</temperature:powerSaved>
[java]     <temperature:id>501</temperature:id>
[java]     <temperature:name>temperature</temperature:name>
[java]     <temperature:long>4.504343</temperature:long>
[java]     <temperature:lat>1.23434</temperature:lat>
[java]     <temperature:humidity>6.6</temperature:humidity>
[java]     <temperature:temp>20.44345</temperature:temp>
[java]   </temperature:sensor>
[java] </temperature:SensorStream>
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] <temperature:SensorStream xmlns:temperature="http://samples.wso2.org">
[java]   <temperature:sensor>
[java]     <temperature:timestamp>19900813115534</temperature:timestamp>
[java]     <temperature:powerSaved>true</temperature:powerSaved>
[java]     <temperature:id>503</temperature:id>
[java]     <temperature:name>temperature</temperature:name>
[java]     <temperature:long>4.504343</temperature:long>
[java]     <temperature:lat>1.23434</temperature:lat>
[java]     <temperature:humidity>6.6</temperature:humidity>
[java]     <temperature:temp>20.44345</temperature:temp>
[java]   </temperature:sensor>
[java] </temperature:SensorStream>
[java]
```

The events received by WSO2 DAS are logged as shown below.

```
[2016-03-09 14:37:31,167] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:true, meta_sensorId:502, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:1.23434, humidity:6.6, sensorValue:20.44345
[2016-03-09 14:37:31,197] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:true, meta_sensorId:501, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:1.23434, humidity:6.6, sensorValue:20.44345
[2016-03-09 14:37:31,201] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:true, meta_sensorId:503, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:1.23434, humidity:6.6, sensorValue:20.44345
```

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 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 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 <DAS_HOME>/samples/cep/producers/http directory.

```
ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsn=0005
```

This builds the HTTP client and publishes the events in the <DAS_HOME>/samples/cep/artifacts/0005/httpReceiver.txt 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
[java] [main] INFO org.wso2.carbon.sample.http.Http - Starting WSO2 Http Client
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] meta_timestamp:19900813115534,
[java] meta_isPowerSaverEnabled:false,
[java] meta_sensorId:100,
[java] meta_sensorName:temperature,
[java] correlation_longitude:20.44345,
[java] correlation_latitude:5.443435,
[java] humidity:8.9,
[java] sensorValue:1.23434
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] meta_timestamp:19900813115534,
[java] meta_isPowerSaverEnabled:false,
[java] meta_sensorId:101,
[java] meta_sensorName:temperature,
[java] correlation_longitude:20.44345,
[java] correlation_latitude:5.443435,
[java] humidity:8.9,
[java] sensorValue:1.23434
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] meta_timestamp:19900813115534,
[java] meta_isPowerSaverEnabled:false,
[java] meta_sensorId:102,
[java] meta_sensorName:temperature,
[java] correlation_longitude:20.44345,
[java] correlation_latitude:5.443435,
[java] humidity:8.9,
[java] sensorValue:1.23434
[java]
```

The events received by WSO2 DAS are logged as shown below.

```
[2016-03-09 15:32:16,194] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:100,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
[2016-03-09 15:32:16,220] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:101,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
[2016-03-09 15:32:16,225] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:102,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
```

Sample 0006 - Receiving Custom Text Events via HTTP Transport

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

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 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 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 <DAS_HOME>/cep/samples/processors/http directory.

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

This builds the HTTP client and publishes the events in the <DAS_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.

```
[echo] Configure -Durl=xxxx and ( -DfilePath=xxxx or -Dsn='sample number')
optionally use -Dusername=xxxx -Dpassword=xxxx
[java] [main] INFO org.wso2.carbon.sample.http.Http - Starting WSO2 Http Client
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] meta:19900813115534,false,100,temperature
[java] correlation:20.44345,5.443435
[java] 8.9,1.23434
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] meta:19900813115534,false,101,temperature
[java] correlation:20.44345,5.443435
[java] 8.9,1.23434
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] meta:19900813115534,false,103,temperature
[java] correlation:20.44345,5.443435
[java] 8.9,1.23434
[java]
```

The events received by WSO2 DAS are logged as shown below.

```
[2016-03-09 19:17:34,609] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:100,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
[2016-03-09 19:17:34,637] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:101,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
[2016-03-09 19:17:34,641] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:103,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
```

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 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 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 <DAS_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 in the <DAS_HOME>/samples/cep/artifacts/0007/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.

```
[echo] Configure -DstreamId=xxxx:x.x.x and ( -Dsn='sample number' or -DfilePath=xxxx ) optionally use -Dprotocol='thrift/binary'
-Dhost=xxxx -Dport=xxxx -Dusername=xxxx -Dpassword=xxxx -Devents=xx -Ddelay='delay between events in ms'
[java] [thrift, localhost, 7611, admin, admin, org.wso2.event.sensor.stream:1.0.0, 0007, , 100, 1000]
[java] Starting WSO2 Event Client
[java] [main] INFO org.wso2.carbon.sample.wso2event.Client - StreamDefinition used :
[java]   "name": "org.wso2.event.sensor.stream",
[java]   "version": "1.0.0",
[java]   "nickName": "",
[java]   "description": "",
[java]   "metaData": [
[java]     {
[java]       "name": "timestamp",
[java]       "type": "LONG"
[java]     },
[java]     {
[java]       "name": "isPowerSaverEnabled",
[java]       "type": "BOOL"
[java]     },
[java]     {
[java]       "name": "sensorId",
[java]       "type": "INT"
[java]     },
[java]     {
[java]       "name": "sensorName",
[java]       "type": "STRING"
[java]     }
[java]   ],
[java]   "correlationData": [
[java]     {
[java]       "name": "longitude",
[java]       "type": "DOUBLE"
[java]     },
[java]     {
[java]       "name": "latitude",
[java]       "type": "DOUBLE"
[java]     }
[java]   ],
[java]   "payloadData": [
[java]     {
[java]       "name": "humidity",
[java]       "type": "FLOAT"
[java]     },
[java]     {
[java]       "name": "sensorValue",
[java]       "type": "DOUBLE"
[java]     }
[java]   ]
[java] }
```

The events received by WSO2 DAS are logged as shown below.

```
[2016-03-10 09:43:09,994] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
[2016-03-10 09:43:10,036] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:501,
meta_sensorName:temperature,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:20.44345
[2016-03-10 09:43:10,139] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:502,
meta_sensorName:temperature,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:20.44345
[2016-03-10 09:43:11,140] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:503,
meta_sensorName:temperature,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:20.44345
[2016-03-10 09:43:12,157] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin disconnected
```

Sample 0008 - Receiving Custom WSO2 Events via WSO2Event Receiver

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

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 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 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 <DAS_HOME>/samples/cep/produce rs/wso2-event directory.

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

This builds the wso2event client and publishes the events in the <DAS_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.

```
[echo] Configure -DstreamId=xxxx:x.x.x and ( -Dsn='sample number' or -DfilePath=xxxx ) optionally use -Dprotocol='thrift/binary'
-Dhost=xxxx -Dport=xxxx -Dusername=xxxx -Dpassword=xxxx -Devents=xx -Ddelay='delay between events in ms'
[java] [thrift, localhost, 7611, admin, admin, org.wso2.event.sensor.stream:1.0.0, 0008, , 100, 1000]
[java] Starting WSO2 Event Client
[java] [main] INFO org.wso2.carbon.sample.wso2event.Client - StreamDefinition used :
[java]   "name": "org.wso2.event.sensor.stream",
[java]   "version": "1.0.0",
[java]   "nickName": "",
[java]   "description": "",
[java]   "metaData": [
[java]     {
[java]       "name": "timestamp",
[java]       "type": "LONG"
[java]     },
[java]     {
[java]       "name": "isPowerSaverEnabled",
[java]       "type": "BOOL"
[java]     },
[java]     {
[java]       "name": "sensorId",
[java]       "type": "INT"
[java]     },
[java]     {
[java]       "name": "sensorName",
[java]       "type": "STRING"
[java]     }
[java]   ],
[java]   "correlationData": [
[java]     {
[java]       "name": "longitude",
[java]       "type": "DOUBLE"
[java]     },
[java]     {
[java]       "name": "latitude",
[java]       "type": "DOUBLE"
[java]     }
[java]   ],
[java]   "payloadData": [
[java]     {
[java]       "name": "humidity",
[java]       "type": "FLOAT"
[java]     },
[java]     {
[java]       "name": "sensorValue",
[java]       "type": "DOUBLE"
[java]     }
[java]   ]
[java] }
```

The events received by WSO2 DAS are logged as shown below.

```
[2016-03-10 10:59:05,546] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
[2016-03-10 10:59:05,612] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: mappedSensorData
Event: meta_timestamp:19900813115534,
meta_id:501,
meta_isPowerSavingMode:false,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
value:20.44345
[2016-03-10 10:59:05,673] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: mappedSensorData
Event: meta_timestamp:19900813115534,
meta_id:502,
meta_isPowerSavingMode:false,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
value:20.44345
[2016-03-10 10:59:06,675] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: mappedSensorData
Event: meta_timestamp:19900813115534,
meta_id:503,
meta_isPowerSavingMode:false,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
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 general [prerequisites required for all samples](#).
2. Navigate to the `<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, follow the instructions in the [Apache ActiveMQ Documentation](#).

3. Configure WSO2 DAS by adding the relevant libraries to [support JMS transport](#).
4. Configure the sample client by adding the relevant jars. For more information, see [setting up JMS for JMS sample clients](#).

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_HOME>/samples/cep/artifacts/0009.
- Creates a stream with the ID org.wso2.event.sensor.stream:1.0.0.
- Creates an event receiver named jmsReceiver.
- Creates an event publisher named jmsLogger to log the received messages.

Executing the sample

1. Wait until the DAS terminal prompts a message similar to the following.

```
[2015-05-17 23:02:55,157] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener} - Connection attempt: 1 for JMS Provider for listener: jmsReceiverMap#topicMap was successful!
[2015-05-17 23:02:55,160] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSTaskManager} - 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.JMSListener} - Started to listen on destination : topicMap of type topic for listener jmsReceiverMap#topicMap
```

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

Then run the following command:

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

It builds the JMS client and publishes the events in the <DAS_HOME>/samples/cep/artifacts/0009/topicMap.csv file to the JMS receiver endpoint.

The events received by WSO2 DAS are logged in the console as shown below.

```
[2015-05-17 23:47:38,339] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: 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
[2015-05-17 23:47:38,340] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:false, meta_sensorId:602, meta_sensorName:temperature, correlation_longitude:90.34344, correlation_latitude:20.44345, humidity:2.3, sensorValue:20.44345
[2015-05-17 23:47:38,340] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:false, meta_sensorId:603, meta_sensorName:temperature, correlation_longitude:90.34344, correlation_latitude:20.44345, humidity:2.3, sensorValue:20.44345
```

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 the JMS 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 incoming events. 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 general [prerequisites required for all samples](#).
2. Navigate to the <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, follow the instructions in the [Apache ActiveMQ Documentation](#).

3. Configure WSO2 DAS by adding the relevant libraries to [support JMS transports](#).
4. Configure the sample client by adding the relevant jars. For more information, see [setting up JMS for JMS sample clients](#).

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_HOME>/samples/cep/artifacts/0010.
- Creates a stream with the ID org.wso2.event.sensor.stream:1.0.0.
- Creates an event receiver named jmsReceiver with custom mapping.
- Creates an event publisher named jmsLogger to log the received messages.

Executing the sample

1. Wait until the DAS terminal prompts a message similar to the following.

```
[2015-05-18 00:01:46,913] INFO {org.wso2.carbon.event.processor.core.internal.CarbonEventManagementService} - Starting polling event adapters
[2015-05-18 00:01:46,916] INFO {org.wso2.carbon.event.input.adapter.core.internal.CarbonInputAdapterRuntime} - Connecting receiver jmsReceiverMap
[2015-05-18 00:01:46,994] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSConnectionFactory} - JMS ConnectionFactory : jmsReceiverMap initialized
[2015-05-18 00:01:47,140] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener} - Connection attempt: 1 for JMS Provider for listener: jmsReceiverMap#topicMap was successful!
[2015-05-18 00:01:47,143] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSTaskManager} - Task manager for event adapter : jmsReceiverMap [re]-initialized
[2015-05-18 00:01:48,146] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener} - Started to listen on destination : topicMap of type topic for listener jmsReceiverMap#topicMap
```

2. Open another terminal and navigate to the <DAS_HOME>/samples/cep/producers/jms/ directory. Then run the following command.

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

It builds the JMS client and publishes the events in the <DAS_HOME>/samples/cep/artifacts/0010/topicMap.csv file to the JMS receiver endpoint.

The events received by WSO2 DAS are logged in the console as shown below.

```
[2015-05-18 00:06:31,083] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
    Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
    meta_isPowerSaverEnabled:false,
    meta_sensorId:501,
    meta_sensorName:temperature,
    correlation_longitude:90.34344,
    correlation_latitude:20.44345,
    humidity:2.3,
    sensorValue:20.44345
[2015-05-18 00:06:31,084] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
    Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
    meta_isPowerSaverEnabled:false,
    meta_sensorId:502,
    meta_sensorName:temperature,
    correlation_longitude:90.34344,
    correlation_latitude:20.44345,
    humidity:2.3,
    sensorValue:20.44345
[2015-05-18 00:06:31,084] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
    Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
    meta_isPowerSaverEnabled:false,
    meta_sensorId:503,
    meta_sensorName:temperature,
    correlation_longitude:90.34344,
    correlation_latitude:20.44345,
    humidity:2.3,
    sensorValue:20.44345
```

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 that adhere to the WSO2Event format via the JMS transport. This sample does not process incoming events. Received events are logged by a log event publisher.

Prerequisites

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

1. Set up the general [prerequisites required for all samples](#).
2. Navigate to the <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, follow the instructions in the [Apache ActiveMQ Documentation](#).

3. Configure WSO2 DAS by adding relevant libraries to [support JMS transport](#).
4. Configure a sample client by adding relevant jars. For more information, see [setting up JMS for JMS sample clients](#).

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_HOME>/

- samples/cep/artifacts/0011.
- Creates a stream with the ID org.wso2.event.sensor.stream:1.0.0.
 - Creates 3 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 DAS terminal prompts a message similar to the following.

```
[2015-05-18 00:31:15,062] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener}
} - Started to listen on destination : topicText of type topic for listener jmsReceiverText#topic
Text
[2015-05-18 00:31:15,062] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener}
} - Started to listen on destination : topicJSON of type topic for listener jmsReceiverJSON#topic
JSON
[2015-05-18 00:31:15,062] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener}
} - Started to listen on destination : topicXML of type topic for listener jmsReceiverXML#topicXM
L
```

2. Open another terminal and navigate to the <DAS_HOME>/samples/cep/producers/jms directory. Then run one of the the following commands depending on the event format in which the event should 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 in the <DAS_HOME>/samples/cep/artifacts/0011/topicMap.csv file to the JMS receiver endpoint.

The events received by WSO2 DAS are logged in the console as shown below.

JSON formatted events:

```
[2015-05-18 00:50:04,592] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:701,
  meta_sensorName:temperature,
  correlation_longitude:90.34344,
  correlation_latitude:20.44345,
  humidity:2.3,
  sensorValue:20.44345
[2015-05-18 00:50:04,594] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:702,
  meta_sensorName:temperature,
  correlation_longitude:90.34344,
  correlation_latitude:20.44345,
  humidity:2.3,
  sensorValue:20.44345
[2015-05-18 00:50:04,595] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:703,
  meta_sensorName:temperature,
  correlation_longitude:90.34344,
  correlation_latitude:20.44345,
  humidity:2.3,
  sensorValue:20.44345
```

Text formatted events:

```
[2015-05-18 00:54:30,999] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:901,
  meta_sensorName:temperature,
  correlation_longitude:20.44345,
  correlation_latitude:5.443435,
  humidity:8.9,
  sensorValue:1.23434
[2015-05-18 00:54:31,000] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:902,
  meta_sensorName:temperature,
  correlation_longitude:20.44345,
  correlation_latitude:5.443435,
  humidity:8.9,
  sensorValue:1.23434
[2015-05-18 00:54:31,000] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:903,
  meta_sensorName:temperature,
  correlation_longitude:20.44345,
  correlation_latitude:5.443435,
  humidity:8.9,
  sensorValue:1.23434
```

for XML formatted events:

```
[2015-05-18 00:56:18,214] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:199008131245,
  meta_isPowerSaverEnabled:true,
  meta_sensorId:801,
  meta_sensorName:temperature,
  correlation_longitude:4.504343,
  correlation_latitude:1.23434,
  humidity:6.6,
  sensorValue:20.44345
[2015-05-18 00:56:18,218] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
  Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:199008131245,
  meta_isPowerSaverEnabled:true,
  meta_sensorId:802,
  meta_sensorName:temperature,
  correlation_longitude:4.504343,
  correlation_latitude:1.23434,
  humidity:6.6,
  sensorValue:20.44345
```

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

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 DAS by adding relevant libraries to [support JMS transport](#).

3. Configure the sample client by adding the relevant jars. See [setting up JMS for JMS sample clients](#).
4. Open the <DAS_HOME>/repository/conf/jndi.properties file and register a connection factory named TopicConnectionFactory by entering the following in the register some connection factories section. (default is the name of the virtually hosted node in Qpid)

```
connectionfactory.TopicConnectionFactory=amqp://admin:admin@clientid/default?brokerlist='tcp://localhost:5672'
By default the jndi.properties file may contain a connection factory named TopicConnectionFactory. You can simply replace it with the following configuration 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 by issuing the following command from the Qpid Broker home. bin\qpid-server

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 DAS terminal prompts a message similar to the following.

```
[2015-05-18 01:31:36,459] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener}
- Started to listen on destination : topicMap of type topic for listener jmsReceiverMap#topicMap
[2015-05-18 01:31:36,459] 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 and navigate to the <CEP_HOME>/samples/cep/producers/jms directory. Then run one of the following commands based on the event format in which the event should 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 in the <DAS_HOME>/samples/cep/artifacts/0012/topicMap.csv and topicXML.txt files to the JMS receiver endpoint.

Events received by DAS are logged in the console as shown below.

Map formatted events:

```
[2015-05-18 01:37:00,000]  INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: 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 [2015-05-18 01:37:00,000]  INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:false, meta_sensorId:602, meta_sensorName:temperature, correlation_longitude:90.34344, correlation_latitude:20.44345, humidity:2.3, sensorValue:20.44345 [2015-05-18 01:37:00,001]  INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:false, meta_sensorId:603, meta_sensorName:temperature, correlation_longitude:90.34344, correlation_latitude:20.44345, humidity:2.3, sensorValue:20.44345
```

XML formatted events:

```
[2015-05-18 01:38:18,327]  INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:199008131245, meta_isPowerSaverEnabled:true, meta_sensorId:801, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:1.23434, humidity:6.6, sensorValue:20.44345 [2015-05-18 01:38:18,330]  INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:199008131245, meta_isPowerSaverEnabled:true, meta_sensorId:802, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:1.23434, humidity:6.6, sensorValue:20.44345
```

Sample 0013 - Receiving Map or Text Events via JMS Transport - WSO2 MB

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

Introduction

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

Prerequisites

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

1. Before you configure the WSO2 DAS:
 - Start WSO2 MB 3.1.0
2. Configure WSO2 DAS by adding relevant libraries to support JMS transport.

3. Configure the sample client by adding relevant jars. For more information, see [setting up JMS for JMS sample clients](#).
4. Open the <DAS_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?bro
```

Building the sample

Start the WSO2 DAS server with the sample configuration numbered **0013**. For more information, see [Starting sample CEP configurations](#).

THE DAS server should be started with a port offset because WSO2 MB is run on the default port 9443. Therefore, append -DportOffset=1 -Dqpid.dest_syntax=BURL to the command as shown below.

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

This sample configuration does the following:

- Changes the default Axis2 repo from <DAS_HOME>/repository/deployment/server to <DAS_HOME>/samples/cep/artifacts/0013.
- Creates a stream with ID org.wso2.event.sensor.stream:1.0.0.
- Creates two event receivers named jmsReceiverMap and jmsReceiverText.
- Creates an event publisher named jmsLogger to log the received messages.
- The ports used by the DAS server are offset by 1 to avoid port conflicts with WSO2 MB.
- The qpid.dest_syntax system property is set to BURL to ensure the usage of the Binding URL(BURL) address syntax.

Executing the sample

1. Wait until the DAS terminal prompts a message similar to the following.

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

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

Run one of the the following commands depending on the event format in which the event should 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 in the <DAS_HOME>/samples/cep/artifacts/0013/topicMap.csv and topicText.txt to the JMS receiver endpoint.

The events received by WSO2 DAS are logged in the console as shown below.

Map formatted events:

```
[2015-05-18 02:02:47,511] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger, Event: 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
[2015-05-18 02:02:47,511] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:602,
meta_sensorName:temperature,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:20.44345
[2015-05-18 02:02:47,512] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:603,
meta_sensorName:temperature,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:20.44345
```

Text formatted events:

```
[2015-05-18 02:04:14,451] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:100,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
[2015-05-18 02:04:14,453] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:101,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
[2015-05-18 02:04:14,454] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:102,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
```

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 adhere to the WSO2 Event format via the SOAP transport. This sample does not process 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 DAS 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 `<DAS_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 `<DAS_HOME>/samples/cep/producers/soap/build.xml` file.

```
[2015-09-15 11:55:02,873] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:4354643, meta_isPowerSaverEnabled:true, meta_sensorId:100, meta_sensorName:data1, correlation_longitude:90.34344, correlation_latitude:5.443435, humidity:8.9, sensorValue:20.44345
[2015-09-15 11:55:02,874] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:4354653, meta_isPowerSaverEnabled:false, meta_sensorId:101, meta_sensorName:data1, correlation_longitude:90.34344, correlation_latitude:5.443435, humidity:8.9, sensorValue:20.44345
[2015-09-15 11:55:02,875] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger, Event: meta_timestamp:4354343, meta_isPowerSaverEnabled:true, meta_sensorId:102, meta_sensorName:data1, correlation_longitude:90.34344, correlation_latitude:5.443435, humidity:8.9, sensorValue:20.44345
```

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 DAS 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 `email Receiver` 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 `cep` as the subject. 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
```

The email events received by the DAS server are logged as shown below.

```
[2015-06-23 17:54:01,468] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: 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
```

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 DAS by adding relevant jars to support [MQTT transport](#).
3. Configure the sample client by adding the relevant jars. For more information, see [setting up MQTT for MQTT sample clients](#).
4. Start the MQTT-supported server. (E.g. Mosquitto)

Building the sample

Start the WSO2 DAS 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 upon successful connection to the MQTT-supported server in the WSO2 DAS logs in

the CLI, navigate to the <DAS_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 <DAS_HOME>/samples/cep/producers/mqtt/build.xml file.

This builds the MQTT client and publishes the events in the <DAS_HOME>/samples/cep/artifacts/0016/sensordata.txt file to the mqttEventReceiver endpoint. The details of the events that are sent are logged as shown below.

```
run:
[echo] Configure -Durl=xxxx -Dtopic=xxxx and (-DfilePath=xxxx and/or -Dsn='sample number')
[java] [main] INFO org.wso2.carbon.sample.mqttclient.MQTTClient - Starting MQTT Client
[java] [main] INFO org.wso2.carbon.sample.mqttclient.MQTTClient - Sending message:
[java] [main] INFO org.wso2.carbon.sample.mqttclient.MQTTClient -
[java] {
[java]     "event": {
[java]         "metaData": {
[java]             "timestamp": 4354643,
[java]             "isPowerSaverEnabled": false,
[java]             "sensorId": 701,
[java]             "sensorName": temperature
[java]         },
[java]         "correlationData": {
[java]             "longitude": 4.504343,
[java]             "latitude": 20.44345
[java]         },
[java]         "payloadData": {
[java]             "humidity": 2.3,
[java]             "sensorValue": 4.504343
[java]         }
[java]     }
[java] }
[java] [main] INFO org.wso2.carbon.sample.mqttclient.MQTTClient - Sending message:
[java] [main] INFO org.wso2.carbon.sample.mqttclient.MQTTClient -
[java] {
[java]     "event": {
[java]         "metaData": {
[java]             "timestamp": 4354643,
[java]             "isPowerSaverEnabled": false,
[java]             "sensorId": 702,
[java]             "sensorName": temperature
[java]         },
[java]         "correlationData": {
[java]             "longitude": 4.504343,
[java]             "latitude": 20.44345
[java]         },
[java]         "payloadData": {
[java]             "humidity": 2.3,
[java]             "sensorValue": 4.504343
[java]         }
[java]     }
[java] }
[java] [main] INFO org.wso2.carbon.sample.mqttclient.MQTTClient - Sending message:
[java] [main] INFO org.wso2.carbon.sample.mqttclient.MQTTClient -
[java] {
[java]     "event": {
[java]         "metaData": {
[java]             "timestamp": 4354643,
[java]             "isPowerSaverEnabled": false,
[java]             "sensorId": 703,
[java]             "sensorName": temperature
[java]         },
[java]         "correlationData": {
[java]             "longitude": 4.504343,
[java]             "latitude": 20.44345
[java]         },
[java]         "payloadData": {
[java]             "humidity": 2.3,
[java]             "sensorValue": 4.504343
[java]         }
[java]     }
[java] }
```

BUILD SUCCESSFUL

The JSON events received by the DAS server are logged as shown below.

```
[2015-06-23 18:41:01,033] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: sensorInfo,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:701,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
[2015-06-23 18:41:01,044] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: sensorInfo,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:702,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
[2015-06-23 18:41:01,051] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: sensorInfo,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:703,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
```

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 do 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 DAS 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 `<Das_Home>/samples/cep/artifacts/0017` directory. Then execute following command to write data to the file.

```
echo  
"timestamp:19900813115534,isPowerSaverEnabled:false,sensorId:103,sensorName:temperatur  
e,longitude:20.44345,latitude:5.443435,humidity:8.9,sensorValue:1.23434" >>  
fileReceiver.txt
```

The received events are logged in the DAS console as shown below.

```
[2015-05-21 19:38:30,230] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.sensor.logger, Event: meta_timestamp:19900813115534, meta_isPowerSaverEnabled:false, meta_sensorId:103, meta_sensorName:temperature, correlation.longitude:20.44345, correlation.latitude:5.443435, humidity:8.9, sensorValue:1.23434
```

Sample 0018 - Receiving JSON Events via Kafka Transport

- Prerequisites
 - Building the sample
 - Executing the sample

Prerequisites

Follow the procedure below to complete the prerequisites needed to execute this sample.

1. Set up the prerequisites required for all samples.
 2. Configure WSO2 DAS 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`. The logs shown in the screenshot below are displayed. For more information, see [Apache Kafka documentation](#).

- Start the Kafka server with the following command.
bin/kafka-server-start.sh config/server.properties
The following logs are displayed

```
[2015-06-24 16:40:16.084] INFO Verifying properties (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.107] INFO Property broker.id is overridden to 0 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.107] INFO Property log.cleaner.enable is overridden to false (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.108] INFO Property log.dirs is overridden to /tmp/kafka-logs (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.108] INFO Property log.retention.check.interval.ms is overridden to 300000 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.108] INFO Property log.retention.hours is overridden to 168 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.108] INFO Property log.segment.bytes is overridden to 1073741824 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.108] INFO Property num.io.threads is overridden to 8 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.108] INFO Property num.network.threads is overridden to 3 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.109] INFO Property num.recovery.threads.per.data.dir is overridden to 1 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.109] INFO Property port is overridden to 9092 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.109] INFO Property socket.receive.buffer.bytes is overridden to 162400 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.109] INFO Property socket.request.max.bytes is overridden to 104857600 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.109] INFO Property socket.send.buffer.bytes is overridden to 162400 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.109] INFO Property zookeeper.connect is overridden to localhost:2181 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.110] INFO Property zookeeper.connection.timeout.ms is overridden to 6000 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:16.133] INFO [Kafka Server 0] Starting to accept connections on localhost:2181 (kafka.server.KafkaServer)
[2015-06-24 16:40:16.134] INFO [Kafka Server 0] Connecting to ZooKeeper at localhost:2181 (kafka.server.KafkaServer)
[2015-06-24 16:40:16.149] INFO Starting zkClient event thread. (org.I0Itec.zkclient.ZkEventThread)
[2015-06-24 16:40:16.144] INFO Client environment:zookeeper.version=3.4.6-1569965, built on 02/28/2014 09:09 GMT (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.144] INFO Client environment:host.name=phenom1 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.144] INFO Client environment:java.version=1.7.0_80 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:java.home=/usr/lib/jvm/java-7-oracle/jre (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:java.class.path=/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..:/core/build/dependant-libbs-2.10.4/*.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..//examples/build/libs/kafka-examples*.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..//contrib/hadoop-consumer/build/libs/kafka-hadoop-consumer*.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..//clients/build/libs/kafka-clients*.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..//libs/jopt-single-3.2.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..//libs/kafka_2.10-0.8.2.1.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/kafka_2.10-0.8.2.1-scalador*.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/kafka_2.10-0.8.2.1-test*.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/kafka-clients-0.8.2.1.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/log4j-1.2.16.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/metrics-core-2.2.0.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/scal-library-2.10.4.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/slf4j-api-1.7.6.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/slf4j-log4j1-2.6.1.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/snappy-java-1.1.6.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/zkclient-3.4.3.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//libs/zookeeper-3.4.6.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..//core/build/libs/kafka_2.10*.jar (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:java.library.path=/usr/java/packages/lib/amd64:/lib64:/lib:/usr/lib (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:java.compiler=-N/A (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:java.endorsed.dirs=/usr/lib/endorsed (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:os.name=Linux (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:os.arch=amd64 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:os.version=3.13.0-57-ef-selinux (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:user.name=thilina (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:user.home=/home/thilina (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.145] INFO Client environment:user.dir=/home/thilina/Software/kafka_2.10-0.8.2.1 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.158] INFO Initiating client connection, connectString=localhost:2181 sessionTimeout=6000 watcher=org.I0Itec.zkclient@6119e2c1 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:16.161] INFO Opening socket connection to server localhost@127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown error) (org.apache.zookeeper.ClientCnxn)
[2015-06-24 16:40:16.161] INFO Socket connection established to localhost@127.0.0.1:2181, initiating session (org.apache.zookeeper.ClientCnxn)
[2015-06-24 16:40:16.166] INFO Session establishment complete on server localhost@127.0.0.1:2181, sessionid = 0x14e2542aac408000, negotiated timeout = 6000 (org.apache.zookeeper.ClientCnxn)
[2015-06-24 16:40:16.167] INFO zookeeper state changed (SyncConnected). (org.I0Itec.zkclient.ZkClient)
[2015-06-24 16:40:16.306] INFO Loading logs. (kafka.log.LogManager)
[2015-06-24 16:40:16.321] INFO Completed load of log sensorInfo-0 with log end offset 0 (kafka.log.Log)
[2015-06-24 16:40:16.329] INFO Completed load of log sensorStream-0 with log end offset 39 (kafka.log.Log)
[2015-06-24 16:40:16.332] INFO Completed load of log sensordata-0 with log end offset 12 (kafka.log.Log)
[2015-06-24 16:40:16.335] INFO Completed load of log filteredSensorStream-0 with log end offset 0 (kafka.log.Log)
[2015-06-24 16:40:16.337] INFO Logs loading complete. (kafka.log.LogManager)
[2015-06-24 16:40:16.337] INFO Starting log cleanup with a period of 300000 ms. (kafka.log.LogManager)
[2015-06-24 16:40:16.339] INFO Starting log flusher with a default period of 9223372036854775807 ms. (kafka.log.LogManager)
[2015-06-24 16:40:16.356] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.Acceptor)
[2015-06-24 16:40:16.357] INFO [Socket Server on Broker 0], Started (kafka.network.SocketServer)
[2015-06-24 16:40:16.397] INFO Will not load MX4J, mx4j-tools.jar is not in the classpath (kafka.utils.MX4JLoader$)
[2015-06-24 16:40:16.428] INFO 0 successfully elected as leader (kafka.server.ZookeeperLeaderElector)
[2015-06-24 16:40:16.603] INFO New leader is 0 (kafka.server.ZookeeperLeaderElector$LeaderChangeListener)
[2015-06-24 16:40:16.650] INFO Registered broker 0 at path /brokers/ids/0 with address phenom1:9092. (kafka.utils.ZkUtils$)
[2015-06-24 16:40:16.657] INFO [Kafka Server 0], started (kafka.server.KafkaServer)
[2015-06-24 16:40:16.755] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions [filteredSensorStream,0],[sensorInfo,0],[sensordata,0],[sensorStream,0] (kafka.server.ReplicaFetcherManager)
[2015-06-24 16:40:16.785] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions [filteredSensorStream,0],[sensorInfo,0],[sensordata,0],[sensorStream,0] (kafka.server.ReplicaFetcherManager)
```

Building the sample

Start the WSO2 DAS 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 `<DAS_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
```

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

This builds the Kafka client and publishes the JSON events in the `<DAS_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.

```

run:
[echo] Configure -Durlxxxxx -topicxxxxx and (-DfilePathxxxx or -Dsm='sample number')
[java] Starting Kafka Client
[java] SLF4J: Class path contains multiple SLF4J bindings.
[java] [main] INFO kafka.utils.VerifiableProperties - Verifying properties
[java] SLF4J: Found binding in [jar:file:/home/thilina/Software/wso2cep-4.0.0-SNAPSHOT/samples/lib/activenq-all-5.7.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
[java] [main] INFO kafka.utils.VerifiableProperties - Property metadata.broker.list is overridden to localhost:9092
[java] SLF4J: Found binding in [jar:file:/home/thilina/Software/wso2cep-4.0.0-SNAPSHOT/samples/lib/slf4j-log4j2-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
[java] [main] INFO kafka.utils.VerifiableProperties - Property serializer.class is overridden to kafka.serializer.StringEncoder
[java] Sending message:
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 701,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
[java] [main] INFO kafka.client.ClientUtils$ - Fetching metadata from broker id:0,host:localhost,port:9092 with correlation id 0 for 1 topic(s) Set(sensorStream)
[java] [main] INFO kafka.producer.SyncProducer - Connected to localhost:9092 for producing
[java] [main] INFO kafka.producer.SyncProducer - Disconnecting from localhost:9092
[java] [main] INFO kafka.producer.SyncProducer - Connected to pheonix:9092 for producing
[java] Sending message:
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 702,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
[java] Sending message:
[java] {
[java]   "event": {
[java]     "metaData": {
[java]       "timestamp": 4354643,
[java]       "isPowerSaverEnabled": false,
[java]       "sensorId": 703,
[java]       "sensorName": temperature
[java]     },
[java]     "correlationData": {
[java]       "longitude": 4.504343,
[java]       "latitude": 20.44345
[java]     },
[java]     "payloadData": {
[java]       "humidity": 2.3,
[java]       "sensorValue": 4.504343
[java]     }
[java]   }
[java] }
[java] [main] INFO kafka.producer.Producer - Shutting down producer
[java] [main] INFO kafka.producer.ProducerPool - Closing all sync producers
[java] [main] INFO kafka.producer.SyncProducer - Disconnecting from pheonix:9092
[java] [main] INFO kafka.producer.Producer - Producer shutdown completed in 20 ms
BUILD SUCCESSFUL

```

The JSON events received by the DAS server are logged in the CLI as shown below.

```

[2015-06-25 08:14:47,555]  INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: sensorInfo,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:702,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
[2015-06-25 08:14:47,556]  INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: sensorInfo,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:703,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343

```

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 DAS 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 `<DAS_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 `<DAS_HOME>/samples/cep/producers/websocket/build.xml` file.

This builds the WebSocket client and publishes the events in the `<DAS_HOME>/samples/cep/artifacts/0019/sensorStreamEvents.txt` file to the `wsReceiver` endpoint. The events sent are logged in the CLI 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

```
[2015-06-24 15:53:49,571] INFO {org.wso2.carbon.event.processor.core.internal.CarbonEventManagementService} - Starting polling event adapters
[2015-06-24 15:53:49,573] INFO {org.wso2.carbon.event.input.adapter.core.internal.InputAdapterRuntime} - Connecting receiver wsReceiver
SLF4J: The requested version 1.6.99 by your slf4j binding is not compatible with [1.5.5, 1.5.6, 1.5.7, 1.5.8, 1.5.9, 1.5.10]
SLF4J: See http://www.slf4j.org/codes.html#version_mismatch for further details.
[2015-06-24 15:53:51,668] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: sensorInfo,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:701,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
[2015-06-24 15:53:51,669] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: sensorInfo,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:702,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
[2015-06-24 15:53:51,670] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: sensorInfo,
Event: meta_timestamp:4354643,
meta_isPowerSaverEnabled:false,
meta_sensorId:703,
meta_sensorName:temperature,
correlation_longitude:4.504343,
correlation_latitude:20.44345,
humidity:2.3,
sensorValue:4.504343
```

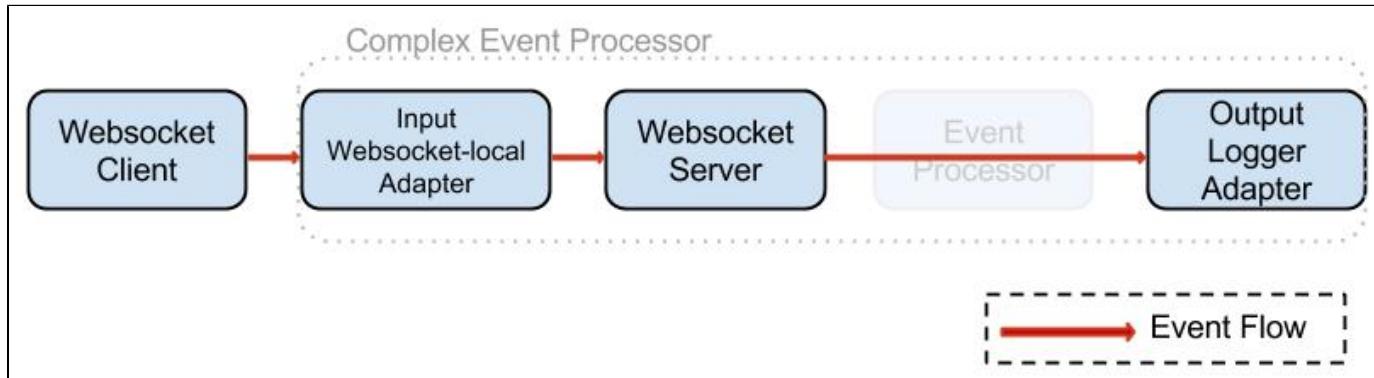
Sample 0020 - Simple JSON Pass-through with WebSocket-Local Input Event Adapter

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

Introduction

This sample demonstrates how the WSO2 DAS can receive events from a WebSocket client and then process them if required.

The following diagram provides an overview of the event flow.



The following is a summary of the steps followed in this sample.

- Step 1: Start the DAS server. This starts an in-built Websocket server in DAS.
 - Step 2: Define an Event Stream in DAS.
 - Step 3: Create an Input Websocket-local Event Adapter in order to allow WSO2 DAS to receive events of the type defined in step 1.
 - Step 4: Create an Output Logger Event Adapter to ensure that events received by WSO2 DAS are logged in the console on which the WSO2 DAS server is run.
 - Step 5: Connect to the DAS-websocket server and publish events to it, using a Websocket client. The published messages are logged in the DAS console.

Note that, for simplicity, we do not set up WSO2 DAS to process the events received by the Websocket-local Input Adapter. Instead of processing events using execution plans, the events are directly sent over to the Output Logger Adapter for logging. This is indicated in the above diagram where the Event Processor component almost

transparent to show that it does not do any processing.

Prerequisites

The following should be completed before running the sample.

1. Set up the prerequisites required for all samples.
2. Navigate to the <DAS_HOME>/samples/cep/utils/input-websocket-local-adapter directory and execute the following command.

```
ant -Dsn=0020
```

This copies the `inputwebsocket.war` webapp to <DAS_HOME>/sample/cep/artifacts/0020/webapps directory.

Starting sample CEP configurations

Start the WSO2 DAS server with the sample configuration numbered **0020**. For instructions, see [Starting sample CEP configurations](#).

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

When these sample configurations are deployed:

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

In other words, all the first four steps listed in Introduction section are completed once the server is started with the sample configuration number **0020**.

The remaining task is to connect to the DAS-websocket server and publish events to it.

Connect to DAS via a Websocket client

In this sample, the web browser is used 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 the DAS 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.

```
>var ws = new WebSocket("ws://localhost:9763/inputwebsocket/wsLocalInputAdapter");
```

Upon successful connection, the output is displayed in the same console as shown below.

```
17:43:14.967 < var ws = new WebSocket("ws://localhost:9763/inputwebsocket/wsLocalInputAdapter");
17:43:14.971 > undefined
17:43:15.021 GET http://localhost:9763/inputwebsocket/wsLocalInputAdapter [HTTP/1.1 101 Switching Protocols 1ms]
```

Now we're ready to publish events to WSO2 DAS.

Step III: Next and final step is to publish three events to WSO2 DAS. The following are the three events to be published.

```
{
  "event": {
    "metaData": {
      "timestamp": 4354643,
      "isPowerSaverEnabled": false,
      "sensorId": 701,
      "sensorName": temperature
    },
    "correlationData": {
      "longitude": 4.504343,
      "latitude": 20.44345
    },
    "payloadData": {
      "humidity": 2.3,
      "sensorValue": 4.504343
    }
  }
}

{
  "event": {
    "metaData": {
      "timestamp": 4354643,
      "isPowerSaverEnabled": false,
      "sensorId": 702,
      "sensorName": temperature
    },
    "correlationData": {
      "longitude": 4.504343,
      "latitude": 20.44345
    },
    "payloadData": {
      "humidity": 2.3,
      "sensorValue": 4.504343
    }
  }
}

{
  "event": {
    "metaData": {
      "timestamp": 4354643,
      "isPowerSaverEnabled": false,
      "sensorId": 703,
      "sensorName": temperature
    },
    "correlationData": {
      "longitude": 4.504343,
      "latitude": 20.44345
    },
    "payloadData": {
      "humidity": 2.3,
      "sensorValue": 4.504343
    }
  }
}
}
```

Issue the following three commands. Each command publishes an event to WSO2 DAS.

```
ws.send("{"event": {"metaData": {"timestamp": 4354643, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": "temperature"}, "correlationData": {"longitude": 4.504343, "latitude": 20.44345}, "payloadData": {"humidity": 2.3, "sensorValue": 4.504343}}});
```

```
ws.send("{"event": {"metaData": {"timestamp": 4354643, "isPowerSaverEnabled": false, "sensorId": 702, "sensorName": "temperature"}, "correlationData": {"longitude": 4.504343, "latitude": 20.44345}, "payloadData": {"humidity": 2.3, "sensorValue": 4.504343}}});
```

```
ws.send("{"event": {"metaData": {"timestamp": 4354643, "isPowerSaverEnabled": false, "sensorId": 703, "sensorName": "temperature"}, "correlationData": {"longitude": 4.504343, "latitude": 20.44345}, "payloadData": {"humidity": 2.3, "sensorValue": 4.504343}}});
```

This is shown in the screenshot below:

The screenshot shows the 'Browser Console' window with the 'JS' tab selected. The console output displays the following log entries:

- 17:58:50.929 var ws = new WebSocket("ws://localhost:9763/inputwebsocket/wsLocalInputAdapter");
- 17:58:50.930 ► undefined
- 17:58:50.980 GET http://localhost:9763/inputwebsocket/wsLocalInputAdapter [HTTP/1.1 101 Switching Protocols 82ms]
- 17:59:13.811 ► ws.send("{"event": {"metaData": {"timestamp": 4354643, "isPowerSaverEnabled": false, "sensorId": 701, "sensorName": "temperature"}, "correlationData": {"longitude": 4.504343, "latitude": 20.44345}, "payloadData": {"humidity": 2.3, "sensorValue": 4.504343}}});
- 17:59:13.812 ► undefined
- 18:01:19.362 ► ws.send("{"event": {"metaData": {"timestamp": 4354643, "isPowerSaverEnabled": false, "sensorId": 702, "sensorName": "temperature"}, "correlationData": {"longitude": 4.504343, "latitude": 20.44345}, "payloadData": {"humidity": 2.3, "sensorValue": 4.504343}}});
- 18:01:19.366 ► undefined
- 18:03:19.039 ► ws.send("{"event": {"metaData": {"timestamp": 4354643, "isPowerSaverEnabled": false, "sensorId": 703, "sensorName": "temperature"}, "correlationData": {"longitude": 4.504343, "latitude": 20.44345}, "payloadData": {"humidity": 2.3, "sensorValue": 4.504343}}});
- 18:03:19.040 ► undefined

Events sent by the browser and received by DAS are logged in the DAS console as shown below.

The screenshot shows the DAS console with the following log entries:

- [2015-05-09 17:59:13.826] INFO (org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter) - Unique ID: sensorInfo, Event: meta_timestamp:4354643, meta_isPowerSaverEnabled:false, meta_sensorId:701, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:20.44345, humidity:2.3, sensorValue:4.504343
- [2015-05-09 18:01:19.369] INFO (org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter) - Unique ID: sensorInfo, Event: meta_timestamp:4354643, meta_isPowerSaverEnabled:false, meta_sensorId:702, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:20.44345, humidity:2.3, sensorValue:4.504343
- [2015-05-09 18:03:19.041] INFO (org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter) - Unique ID: sensorInfo, Event: meta_timestamp:4354643, meta_isPowerSaverEnabled:false, meta_sensorId:703, meta_sensorName:temperature, correlation_longitude:4.504343, correlation_latitude:20.44345, humidity:2.3, sensorValue:4.504343

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 that adhere to the WSO2 Event format via the JMS transport using JMS Queue. Here we do not do any processing on the incoming event. A log event publisher logs the received events allowing them to be verified.

Prerequisites

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

1. Start the Apache ActiveMQ server.

This guide uses ActiveMQ versions 5.7.0 - 5.9.0. If you want to use a later version, follow the instructions with regard to the required configuration changes in the [Apache ActiveMQ Documentation](#).

2. Configure WSO2 DAS by adding relevant libraries to [support JMS transport](#).
3. Configure the sample client by adding relevant jars. For more information, see [Setting up JMS for JMS sample clients](#).

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_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 DAS terminal prompts a message similar to the following.

```
[2015-07-15 11:24:52,264] INFO {org.wso2.carbon.event.input.adapter.core.internal.InputAdapterRuntime}
- Connecting receiver jmsReceiverMap
[2015-07-15 11:24:52,354] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSConnectionFactory}
- JMS ConnectionFactory : jmsReceiverMap initialized
[2015-07-15 11:24:52,513] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener}
- Connection attempt: 1 for JMS Provider for listener: jmsReceiverMap#queueMap was successful!
[2015-07-15 11:24:52,517] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSTaskManager}
- Task manager for event adapter : jmsReceiverMap [re-]initialized
[2015-07-15 11:24:53,519] INFO {org.wso2.carbon.event.input.adapter.jms.internal.util.JMSListener}
- Started to listen on destination : queueMap of type queue for listener jmsReceiverMap#queueMap
```

2. Open another terminal and navigate to <DAS_HOME>/samples/cep/producers/jms.

Run the following command. Change the event format depending on the format in which the event should be published:

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

It builds the jms client and publishes the events in <DAS_HOME>/samples/cep/artifacts/0021/queueMap.csv to the JMS receiver endpoint.

The events received by WSO2 DAS for Map formatted events are logged in the CLI as shown below.

```
[2015-07-15 11:27:48,168] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: jmsLogger,
  Event: 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
[2015-07-15 11:27:48,169] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: jmsLogger,
  Event: meta_timestamp:19900813115534,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:602,
  meta_sensorName:temperature,
  correlation_longitude:90.34344,
  correlation_latitude:20.44345,
  humidity:2.3,
  sensorValue:20.44345
[2015-07-15 11:27:48,169] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: jmsLogger,
  Event: meta_timestamp:19900813115534,
  meta_isPowerSaverEnabled:false,
  meta_sensorId:603,
  meta_sensorName:temperature,
  correlation_longitude:90.34344,
  correlation_latitude:20.44345,
  humidity:2.3,
  sensorValue:20.44345
```

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 a specified event stream. 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](#). Also, there are some of the Regular Expressions that are used to classify the text that 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 that are 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
 - a. timestamp: \[(.+)\]
 - b. information type: \]\s+(\w*)\s+
 - c. class name: \{(.+)\}
 - d. 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)
 - a. first element: ([^,]+)
 - b. second element: [^,]+, ([^,]+)
 - c. third element: [^,]+, [^,]+, ([^,]+)

Prerequisites

Set up the [prerequisites](#) required for all samples.

Building the sample

Start the WSO2 DAS 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 `fileTailerReceiv er`.

- An event publisher named `fileLogger` to log the messages.

Executing the sample

Navigate to the `<Das_Home>/samples/cep/artifacts/0022/fileReceiver.csv` file, and append several lines in a format similar to the first line that 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 the command given below to add the lines to the `fileReceiver.csv`. Do not edit the file directly and save because that does not allow the file to be successfully tailed.

```
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. The input values mapped by the receiver are logged by the logger.

The events received by the DAS server are logged in the CLI as shown below.

```
[2015-07-13 09:55:21,256] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.sensor.logger,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:103,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23434
[2015-07-13 09:55:21,258] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} -
Unique ID: org.wso2.event.sensor.logger,
Event: meta_timestamp:19900813115534,
meta_isPowerSaverEnabled:false,
meta_sensorId:103,
meta_sensorName:temperature,
correlation_longitude:20.44345,
correlation_latitude:5.443435,
humidity:8.9,
sensorValue:1.23444
```

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:

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

The above query does the following.

- Processes the events received through the `sensor_stream` 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 these events as output events through the `sensor_value_projected_stream` stream..

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

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 DAS server with the sample configuration numbered **0101**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to `<DAS_HOME>/samples/cep/artifacts/0101` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`).

Executing the sample

1. Open a terminal and navigate to the <DAS_HOME>/samples/cep/producers/wso2-event directory. Issue the following command from this location.

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

It builds and runs the wso2event producer that sends sensor data to the DAS server.

2. View the output events received from the DAS console. This sample uses the logger adaptor to log output events to the console.

The output of the consumer sending events from the producer is logged as follows.

```
[2015-05-13 14:39:32,778] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
[2015-05-13 14:39:32,811] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: Projected Sensor Values, Event: meta_sensorId:501,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
value:20.44345
[2015-05-13 14:39:33,481] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: Projected Sensor Values, Event: meta_sensorId:502,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
value:20.44345
[2015-05-13 14:39:34,482] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: Projected Sensor Values, Event: meta_sensorId:503,
correlation_longitude:90.34344,
correlation_latitude:20.44345,
humidity:2.3,
value:20.44345
[2015-05-13 14:39:35,496] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin disconnected
```

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 enrich events with new attributes. This sample uses the Event Simulator for inputs and the logger publisher for logging the outputs to the DAS console.

The execution plan used in this sample is 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 does the following.

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

The second query does the following.

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

The third query does the following

- Processes the events received through the `TempStream` stream.
- When selecting the attributes, concatenates `roomNo` and `deviceId` 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 `scale` with the value `F` under the `select` clause for each event received.
- Emits these events as output events through the `TransformedRoomTempStream` stream.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 DAS server with the sample configuration numbered **0102**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to <DAS_HOME>/samples/cep/artifacts/0102 (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Log into the DAS Management Console.
2. Go to **Tools -> Event Simulator**. The events.csv file that contains some sample data is displayed in the **Send multiple events** section. Click **Play** to start sending sample events from the file. Here, you can perform other operations such as **Pause**, **Stop** and **Resume** if required. The output events received are logged in the DAS console as shown below.. This sample uses the logger adaptor to log output events to the console.

```
[2015-07-02 14:22:06,716] INFO {org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to the junction. Stream:TempStream:1.0.0
[2015-07-02 14:22:06,722] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: EnrichedRoomTempStreamLogger,
Event: {"event":{"payloadData":{"roomNo":275,"temp":25.67,"scale":"C"}}}
[2015-07-02 14:22:06,723] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: TransformedRoomTempStreamLogger,
Event: uid:275-1259283,
temp:78.206,
scale:F
[2015-07-02 14:22:07,720] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: EnrichedRoomTempStreamLogger,
Event: {"event":{"payloadData":{"roomNo":972,"temp":26.3,"scale":"C"}}}
[2015-07-02 14:22:07,721] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: TransformedRoomTempStreamLogger,
Event: uid:972-2458199,
temp:79.34,
scale:F
[2015-07-02 14:22:08,721] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: EnrichedRoomTempStreamLogger,
Event: {"event":{"payloadData":{"roomNo":175,"temp":29.3,"scale":"C"}}}
[2015-07-02 14:22:08,721] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: TransformedRoomTempStreamLogger,
Event: uid:175-9872343,
temp:84.74000000000001,
scale:F
[2015-07-02 14:22:09,722] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: EnrichedRoomTempStreamLogger,
Event: {"event":{"payloadData":{"roomNo":276,"temp":19.3,"scale":"C"}}}
[2015-07-02 14:22:09,722] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: TransformedRoomTempStreamLogger,
Event: uid:276-9812232,
temp:66.74000000000001,
scale:F
[2015-07-02 14:22:10,723] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: EnrichedRoomTempStreamLogger,
Event: {"event":{"payloadData":{"roomNo":294,"temp":40.2,"scale":"C"}}}
[2015-07-02 14:22:10,724] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: TransformedRoomTempStreamLogger,
Event: uid:294-2083739,
temp:104.36000000000001,
scale:F
```

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 do 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 DAS console.

The execution plan used in this sample is as follows:

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

The first query does the following.

- Processes the events received through the TempStream stream.
- Checks for the condition roomNo > 245 and roomNo <= 365 and temp > 40 inside the filter.

- If the condition is true,
 - Selects the attributes (i.e., roomNo, temp) specified under the select clause, from each event received.
 - Emits these events as output events through the AlertServerRoomTempStream stream.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 DAS server with the sample configuration numbered **0103**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to <DAS_HOME>/samples/cep/artifacts/0103 (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Log into the DAS Management Console which is located at `https://<DAS_HOST>:<DAS_PORT>/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. Output events received from the DAS console are logged as follows. This sample uses the logger adaptor to log output events to the console.

```
[2015-07-02 15:43:23,504] INFO {org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to the junction. Stream:TempStream:1.0.0
[2015-07-02 15:43:27,511] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AlertLogger,
Event: roomNo:294,
temp:40.2
```

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 DAS console. 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](#).

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

```
-- 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 does the following

- 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. The average of the temp values is named as avgTemp.
- The all events clause in the insert statement makes the query to be triggered by both current events and expired events (current events are the incoming events to the window. An expired event is an event emitted by the window after being kept for 1 minute).
- Emits these 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 the arrival/expiration of each incoming event.

The second query,

- Processes the events received through the TempStream.
- Defines a time batch window of 1 minute to keep all incoming events and then emit 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. The average of the temp values is named as avgTemp.
- The all events clause in the insert statement makes the query to be triggered by both current events and expired events (current events are the incoming events to the window. An expired event is an event emitted by the window after being kept in the window for 1 minute.)
- 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 every 1 minute.

Another execution plan continuously calculates the average temperature from the beginning. It includes the following queries.

```
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 DAS server with the sample configuration numbered **0104**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to <DAS_HOME>/samples/cep/artifacts/0104 (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Log into the DAS Management Console via the `https://<DAS_HOST>:<DAS_PORT>/carbon` URL.
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. View the output events received from the DAS 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.

The output of events sent by the consumer from the producer are logged in the CLI as shown in the following example.

```
[2015-07-02 16:34:56,931] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgTempFromStartStreamLogger, Event: {"event":{"payloadData":{"roomNo":294,"avgTemp":27.939230769230768}}}
[2015-07-02 16:34:57,932] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgRoomTemp, Event: roomNo:175, avgTemp:26.6
[2015-07-02 16:34:57,933] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgTempFromStartStreamLogger, Event: {"event":{"payloadData":{"roomNo":175,"avgTemp":27.665}}}
[2015-07-02 16:34:58,934] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgRoomTemp, Event: roomNo:275, avgTemp:24.458571428571428
[2015-07-02 16:34:58,934] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgTempFromStartStreamLogger, Event: {"event":{"payloadData":{"roomNo":275,"avgTemp":27.354}}}
[2015-07-02 16:34:59,384] INFO {org.wso2.carbon.event.processor.manager.core.internal.CarbonEventManagementService} - Starting polling event adapters
[2015-07-02 16:35:19,841] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgRoomTempEveryOneMin, Event: roomNo:275, avgTemp:24.458571428571428
[2015-07-02 16:35:19,841] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgRoomTempEveryOneMin, Event: roomNo:175, avgTemp:26.6
[2015-07-02 16:35:19,841] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgRoomTempEveryOneMin, Event: roomNo:294, avgTemp:35.366666666666667
[2015-07-02 16:35:44,917] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgRoomTemp, Event: roomNo:275, avgTemp:24.256666666666667
[2015-07-02 16:35:45,917] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: AvgRoomTemp, Event: roomNo:175,
```

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 DAS console.

The execution plan used in this sample is 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 does the following.

- Processes the events received through the `TempStream`, filters events with `temp > 30.0` condition 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 `isOn == false` condition, 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 creates 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 an 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 DAS server with the sample configuration numbered **0105**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to <DAS_HOME>/samples/cep/artifacts/0105 (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Log into the DAS management console that is located at https://<DAS_HOST>:<DAS_PORT>/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 named RegularStream:1.0.0.
- Click **Play** for both files to start sending sample events in these files.
3. View the output events received from the DAS console. This sample uses the logger adaptor to log output events to the console.

The output of the consumer sending events from the producer is logged as shown in the following example.

```
[2015-07-02 16:57:08,152] INFO {org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to the junction. Stream:TempStream:1.0.0
[2015-07-02 16:57:16,931] INFO {org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to the junction. Stream:RegulatorStream:1.0.0
[2015-07-02 16:57:18,936] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: RegulatorActionLogger,
Event: roomNo:294,
temp:40.2,
deviceID:LNMNJAJS,
action:start
```

Sample 0106 - Using in-memory 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 make use of an in-memory event table to identify blacklisted transactions. This sample uses the Event Simulator for inputs and the logger publisher for logging the outputs to the DAS console.

The execution plan used in this sample is as follows:

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

Given above is the table definition. It does the following.

- 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 the `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.

```
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 `WhiteListPurchaseStream`.

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

The last query is used to clean up the table from an external trigger event through the `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 the prerequisites required for all samples.

Building the sample

Start the WSO2 DAS server with the sample configuration numbered **0106**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to <DAS_HOME>/samples/cep/artifacts/0106 (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Log into the DAS Management Console.
2. Go to **Tools -> Event Simulator**. Under the **Multiple Events** section, 4 files with sample data are listed as shown below.

[Send multiple events](#)

Input Data by File switch to configure database for simulation			
File	Stream Configuration	Delay between events(ms)	Action
deleteUserEvents.csv	DeleteAllUsers:1.0.0	1000	Play Configure Delete
purchaseEvents.csv	PurchaseStream:1.0.0	1000	Play Configure Delete
userEvents.csv	CardUserStream:1.0.0	1000	Play Configure Delete
blackListUserEvents.csv	BlackListStream:1.0.0	1000	Play Configure Delete
Choose File No file chosen		upload	

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` file contains credit card transactions data. Click **Play** to send the transaction data.
6. After sending sample events from the `purchaseEvents.csv` file, you can view the output as shown below.

```
Event: cardNo:4012-3253-3456-1881,
name:Srinath,
price:123.45
[2015-07-14 17:57:23,442] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger,
Event: cardNo:4012-3253-5632-6432,
name:Sachini,
price:123.56
[2015-07-14 17:57:24,442] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger,
Event: cardNo:4012-5678-2345-1234,
name:Mohan,
price:345.5
[2015-07-14 17:57:25,443] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger,
Event: cardNo:4012-3253-5632-6432,
name:Sachini,
price:320.0
```

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 the Event Simulator for inputs and the logger publisher for logging the outputs to the DAS console.

The execution plan used in this sample are as follows:

```
@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. It does the following:

- 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 DAS. If needed you can define a separate data source pointing to some other DB such as MySQL and use it here.)

```
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 is 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 are joined based on the card number.
- Emits those events as output events through the `TransformedRoomTempStream`.

```
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 DAS server with the sample configuration numbered **0107**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to `<DAS_HOME>/samples/cep/artifacts/0107` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`).

Executing the sample

1. Log into the DAS Management Console using the following URL and `admin/admin` credentials: `https://<DAS_HOST>:<DAS_PORT>/carbon`.
2. Click **Tools**, and then click **Event Simulator**. You view four files listed under the **Send multiple Events** section that contains some sample data as follows.

Send multiple events

Input Data by File 			
File	Stream Configuration	Delay between events(ms)	Action
<code>deleteUserEvents.csv</code>	<code>DeleteAllUsers:1.0.0</code>	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<code>purchaseEvents.csv</code>	<code>PurchaseStream:1.0.0</code>	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<code>userEvents.csv</code>	<code>CardUserStream:1.0.0</code>	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<code>blackListUserEvents.csv</code>	<code>BlackListStream:1.0.0</code>	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>		

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. The output logs can be viewed in the CLI in which you run the products as shown below.

```
[2015-09-21 17:38:34,884] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger, Event: cardNo:4012-3253-3456-1881, name:Srinath, price:123.45
[2015-09-21 17:38:36,883] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger, Event: cardNo:4012-3253-5632-6432, name:Sachini, price:123.56
[2015-09-21 17:38:37,883] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger, Event: cardNo:4012-5678-2345-1234, name:Mohan, price:345.5
[2015-09-21 17:38:38,884] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger, Event: cardNo:4012-3253-5632-6432, name:Sachini, price:320.0
[2015-09-21 17:39:30,831] INFO {org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to the junction. Stream:DeleteAllUsers:1.0.0
```

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 DAS console. 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](#).

The query used in this sample is as follows:

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

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

The query does the following.

- Processes the events received through the `atmStatsStream`.
- First looks for an event `a1` where the `amountWithdrawn` is less than 100.
- Next it looks for another event `b1` where the card number is the same as previous and the `amountWithdrawn` is greater than 10000. The `within` keyword ensures that this condition should be satisfied within a day.
- When the pattern condition is satisfied, selects the attribute and renames them as:
`a1.cardNo as cardNo, a1.cardHolderName as cardHolderName, b1.amountWithdrawn as amountWithdrawn, 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 DAS server with the sample configuration numbered **0108**. For instructions, see [Starting sample](#)

CEP configurations.

This sample configuration points the default Axis2 repo to <DAS_HOME>/samples/cep/artifacts/0108 (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Log into the DAS Management Console.
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. View the output events received from the DAS console. This sample uses the logger adaptor to log output events to the console.

The following is an example screenshot of the output of the consumer sending events from the producer:

```
[2015-07-03 12:28:44,070] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: PossibleFraudStreamLogger,
Event: cardNo:ED936784773,
cardHolderName:Mohan,
amountWithdrawn:15000.0,
Location>New York,
cardHolderMobile:9667339388
```

Sample 0109 - Detecting trends with sequences

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 outputs to the DAS 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 where the price is 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 a 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 DAS server with the sample configuration numbered **0109**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to `<DAS_HOME>/samples/cep/artifacts/0109` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`).

Executing the sample

1. Log into the DAS Management Console.
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. View the output events received from the DAS console. This sample uses the logger adaptor to log output events to the console.

The following is an example screenshot of the output of the consumer sending events from the producer:

```
[2015-07-03 13:45:44,878] INFO {org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to the junction. Stream:StockStream:1.0.0
[2015-07-03 13:45:50,886] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: PeakStreamLogger,
Event: priceInitial:29.6,
pricePeak:57.6,
priceAfterPeak:47.6
```

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 do 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:

```
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 takes 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 DAS server with the sample configuration numbered **0110**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to `<DAS_HOME>/samples/cep/artifacts/0110` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`).

Executing the sample

1. Log into the DAS Management Console.
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 DAS console. This sample uses the logger adaptor to log output events to the console.

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

```
[2015-07-03 13:54:25,653] INFO {org.wso2.carbon.event.stream.core.internal.EventJunction} - Producer added to the junction. Stream:StockStream:1.0.0
[2015-07-03 13:54:31,661] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: PeakStreamLogger,
Event: symbol:WS02,
priceInitial:29.6,
pricePeak:57.6,
priceAfterPeak:47.6
[2015-07-03 13:54:32,662] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: PeakStreamLogger,
Event: symbol:FB,
priceInitial:45.6,
pricePeak:45.9,
priceAfterPeak:45.0
```

Sample 0111 - Detecting non-occurrences with Patterns

[Introduction](#)

[Prerequisites](#)

[Building the sample](#)

[Executing the sample](#)

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).

The queries used in this sample are as follows.

```

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 publishes it to the `overdue_deliveries_stream` event stream. This event stream is then used in the second query.

The second query uses a pattern. It looks for a pattern where an arrival event (from the `arrivals` stream event stream) is always followed by a delivery event (from the `deliveries` stream event stream), or by an overdue delivery event (i.e., an arrival event that is 2 minutes old from the `overdue_deliveries` stream event stream). 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.

The third query is used to detect whether the delivery event is null where the `deliveryId` is null if the delivery event is null.

Prerequisites

For general prerequisites, see [CEP Samples Setup](#). (The `apache-axiom.jar` needs to be copied to the `<DAS_HOME>/samples/cep/lib` directory in order to send XML event types.)

Building the sample

Start the WSO2 DAS server with the sample configuration numbered **0111**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to `<DAS_HOME>/samples/cep/artifacts/0111` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`).

Executing the sample

1. Open a new terminal, go to `<DAS_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.

```
ant -Dsn=0111
```

2. View the logs in the DAS server when logger service is deploying. For example, after deployment, the Web app is able to receive messages sent from the DAS server.
3. Navigate to <DAS_HOME>/samples/cep/producers/http and execute the following command.

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

This reads the `arrivalEvents.txt` file that contains a few sample arrival events and then sends them to the specified URL.

4. Execute the following command:

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

This reads the `deliveryEvents.txt` file that contains some sample delivery events and sends them to the specified URL.

The output is logged in the console as shown below after 2 minutes.

```
Event received : trackingId:1002,
customerName:Mike,
telephoneNo:650-720-8600
Event received : trackingId:1005,
customerName:Steve,
telephoneNo:420-100-2000
```

Sample 0112 - Analyzing Twitter Feeds Using Partitions

- [Introduction](#)
- [Prerequisites](#)
- [Building the sample](#)
- [Executing the sample](#)

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 DAS 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 `<DAS_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 `<DAS_HOME>/samples/cep/producers/wso2-event/build.xml` file.

This builds and runs the WSO2Event producer that sends Twitter feed data to the DAS server. The details of the sent events are logged as shown below.

```

run:
[echo] Configure -DstreamId=xxxx:x.x.x and (-Dsn='sample number' or -DfilePath=xxxxx ) optionally use -Dprotocol='thrift/binary' -Dhost=xxxx -Dport=xxxx -Dusername=xxxx -Dpassword=xxxx -Devents=xx -Ddelay='delay between events'
in ms
[echo]
[echo] Starting WSO2 Event Client
[java] [main] INFO org.wso2.carbon.sample.wso2event.Client - StreamDefinition used :{
[java]   "name": "twitterFeed",
[java]   "version": "1.0.0",
[java]   "payloadData": [
[java]     {
[java]       "name": "company",
[java]       "type": "STRING"
[java]     },
[java]     {
[java]       "name": "wordCount",
[java]       "type": "INT"
[java]     }
[java]   ]
[java] }
```

Details of the events received by the DAS server are logged in the CLI as shown below.

```

2015-05-07 09:48:06,116] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
2015-05-07 09:48:09,895] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: highFrequentTweetsLogger, Event: {"event":{"payloadData":{"company":"FB","words":14}}}
2015-05-07 09:48:11,892] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: highFrequentTweetsLogger, Event: {"event":{"payloadData":{"company":"WSO2","words":16}}}
2015-05-07 09:48:13,890] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: highFrequentTweetsLogger, Event: {"event":{"payloadData":{"company":"FB","words":26}}}
```

Sample 0113 - Limiting the Output Rate of an Event Stream

- [Introduction](#)
- [Prerequisites](#)

- Building the sample
- Executing the sample

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. Custom events are emitted every 5 events. 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](#).

```
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 DAS server with the sample configuration numbered **0113**. For instructions, see [Starting sample CEP configurations](#). This sample configuration creates the following.

- Two streams named `org.wso2.sample.login.info:1.0.0` and `org.wso2.sample.out.unique.login: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 `<DAS_HOME>/samples/cep/producers/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 `<DAS_HOME>/samples/cep/producers/http/build.xml` file.

This builds the HTTP client and publishes the events defined in the `<DAS_HOME>/samples/cep/artifacts/0113/loginInfoReceiver.txt` file to the `loginInfoReceiver` endpoint. The details of the events that are sent are logged as shown below.

```

run:
[echo] Configure -Durl=xxxx and (-DfilePath=xxxx or -Dsn='sample number') optionally use -Dusername=xxxx -Dpassword=xxxx
[java] Starting WSO2 Http Client
[java] Sending message:
[java]
[java] <events>
[java]   <event>
[java]     <payloadData>
[java]       <username>ann@org1.com</username>
[java]       <ipAddress>192.168.1.100</ipAddress>
[java]     </payloadData>
[java]   </event>
[java] </events>
[java] Sending message:
[java]
[java] <events>
[java]   <event>
[java]     <payloadData>
[java]       <username>jeff@wso2.com</username>
[java]       <ipAddress>192.168.1.23</ipAddress>
[java]     </payloadData>
[java]   </event>
[java] </events>
[java] Sending message:
[java]
[java] <events>
[java]   <event>
[java]     <payloadData>
[java]       <username>john@org3.com</username>
[java]       <ipAddress>192.168.1.240</ipAddress>
[java]     </payloadData>
[java]   </event>
[java] </events>
[java] Sending message:
[java]
[java] <events>
[java]   <event>
[java]     <payloadData>
[java]       <username>gavin@wso2.org</username>
[java]       <ipAddress>192.168.0.2</ipAddress>
[java]     </payloadData>
[java]   </event>
[java] </events>
[java] Sending message:
[java]
[java] <events>
[java]   <event>
[java]     <payloadData>
[java]       <username>lang@wso2.o</username>
[java]       <ipAddress>10.8.8.23</ipAddress>
[java]     </payloadData>
[java]   </event>
[java] </events>
[java] Sending message:
[java]
[java] <events>
[java]   <event>
[java]     <payloadData>
[java]       <username>sam@org2.com</username>
[java]       <ipAddress>192.248.8.68</ipAddress>
[java]     </payloadData>
[java]   </event>
[java] </events>
[java]

BUILD SUCCESSFUL
Total time: 1 second

```

The output events received by the DAS server are logged as shown below.

```
[2015-05-19 11:15:12,526] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: Unique IP, Event: 10.8.8.23
```

Sample 0114 - Using External Time Windows

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

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:

```

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](#).

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/samples/cep/artifacts/0114 (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Open a terminal, go to <DAS_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 DAS 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

When events are sent from the producer, the output from the DAS console can be viewed as shown below. The output is logged by the logger publisher that is used in this sample.

```

[2015-05-14 16:30:13,151] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
[2015-05-14 16:30:14,859] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: alert, Event: cardNumber:1234,
transactionCount:2,
totalAmount:1010.25
[2015-05-14 16:30:16,860] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: alert, Event: cardNumber:5678,
transactionCount:2,
totalAmount:260.0
[2015-05-14 16:30:17,893] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin disconnected

```

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](#).

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/samples/cep/artifacts/0115 (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Open another terminal, go to <DAS_HOME>/samples/cep/producers/http and run the following command:

```
ant -Durl=http://localhost:9763/endpoints/buildStatisticsEventReceiver -Dsn=0115
```

It builds the http client and publishes the events at <DAS_HOME>/samples/cep/artifacts/0115/buildStatisticsEvents.txt to the buildStatisticsEventReceiver http endpoint.

2. You can see the events getting received by DAS by the logs in its console.

```
[2016-07-12 17:25:00,013] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: Build Failure,
Event: {"event": {"payloadData": {"project": "Carbon Analytics", "productTeam": "CEP", "lastCommitter": "Suho", "totalBuildFailures": 1}}}
[2016-07-12 17:25:00,013] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: Build Failure,
Event: {"event": {"payloadData": {"project": "Carbon Mediation", "productTeam": "Carbon", "lastCommitter": "Sameera", "totalBuildFailures": 1}}}
```

The above query will output the processed events at "1 am" as defined in the execution plan. You can edit the query in "BuildFailureStatisticsPlan.siddhiql" file which is located in the <DAS_HOME>/samples/cep/a

rtifacts/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](#).

```
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 DAS 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:

```
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 DAS 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 <DAS_HOME>/samples/cep/artifacts/0116 directory (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Log into the DAS management console.
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 DAS 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.

```
strikeouts:32,
errors:3,
stderr:815.6194361796848,
beta0:0.0,
beta1:0.0,
beta2:0.0,
beta3:0.0,
beta4:0.0
[2016-07-12 13:45:26,926] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: regression,
Event: salary:200,
rbi:33,
walks:14,
strikeouts:96,
errors:6,
stderr:716.093433277225,
beta0:0.0,
beta1:36.80862538094903,
beta2:0.0,
beta3:0.0,
beta4:0.0
[2016-07-12 13:45:28,929] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: regression,
Event: salary:140,
rbi:22,
walks:19,
strikeouts:56,
errors:3,
stderr:607.2738094306985,
beta0:0.0,
beta1:37.480339598083994,
beta2:0.0,
beta3:0.0,
beta4:0.0
[2016-07-12 13:45:30,931] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: regression,
Event: salary:115,
rbi:2,
walks:4,
strikeouts:3,
errors:0,
stderr:558.8420971185277,
beta0:0.0,
beta1:37.251279144927615,
beta2:0.0,
beta3:0.0,
beta4:0.0
[2016-07-12 13:45:31,953] INFO {org.wso2.carbon.event.processor.manager.core.internal.CarbonEventManagementService} - Starting polling event receivers
[2016-07-12 13:45:32,933] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: regression,
Event: salary:1907,
rbi:73,
walks:50,
strikeouts:100,
errors:14,
stderr:525.0470795185812,
beta0:0.0,
beta1:38.233299938571875,
beta2:0.0,
beta3:0.0,
beta4:0.0
```

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,

```
from          ServerLogs[meta_tenantId      !=      -1 2 3 4]
select        meta_timestamp,           meta_tenantId,      className,      logType
insert into TenantLogs;

from          ServerLogs[logType=='WARN'      or      logType=='warn']
select        meta_timestamp,           meta_tenantId,      className,      msg
insert into WarnTypeLogStream;
```

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.

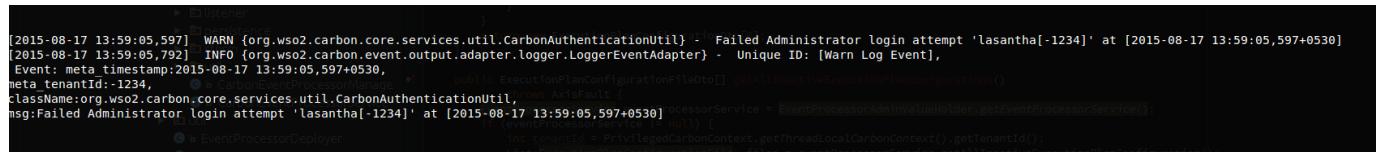
Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/samples/cep/artifacts/0117 directory (by default, the Axis2 repo is <DAS_HOME>/repository/deployment/server).

Executing the sample

1. Log into the DAS management console. Confirm that all the artifacts are deployed properly.
2. Log out and enter incorrect credentials to log into the DAS management console. This will generate a WARN log entry.
3. See the output events received from the DAS console. This sample uses the logger adaptor to log output events to the console.



```
[2015-08-17 13:59:05,597]  WARN {org.wso2.carbon.core.services.util.CarbonAuthenticationUtil} - Failed Administrator login attempt 'lasantha[-1234]' at [2015-08-17 13:59:05,597+0530]
[2015-08-17 13:59:05,792]  INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: [Warn Log Event], Event: meta_timestamp:2015-08-17 13:59:05,597+0530, meta_tenantId:-1234, CarbonEventProcessorUsage: public ExecutionPlanConfigurationFileEditor() throws ExecutionPlanConfigurationException(), className:org.wso2.carbon.core.services.util.CarbonAuthenticationUtil, msg:Failed Administrator login attempt 'lasantha[-1234]' at [2015-08-17 13:59:05,597+0530]ProcessorService + EventProcessorAdminImpl@10.10.10.10/getEventProcessorService(), EventProcessorDeployer
```

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 DAS console.

The execution plan used in this sample are as follows:

```
@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.

```
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 is 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 are 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 exists, deletes all the records in the `CardUserTable`.

Prerequisites

Set up prerequisites required for all samples.

Building the sample

Start the WSO2 DAS 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 `<DAS_HOME>/samples/cep/artifacts/0118` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`).

Executing the sample

- Log into the DAS management console.
- In the **Tools** tab, click **Event Simulator** to open the **Event Stream Simulator** page. In the **Send multiple events** section, 4 files that contain sample data are listed as follows.

Send multiple events

Input Data by File  switch to configure database for simulation			
File	Stream Configuration	Delay between events(ms)	Action
<code>deleteUserEvents.csv</code>	<code>DeleteAllUsers:1.0.0</code>	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<code>purchaseEvents.csv</code>	<code>PurchaseStream:1.0.0</code>	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<code>userEvents.csv</code>	<code>CardUserStream:1.0.0</code>	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<code>blackListUserEvents.csv</code>	<code>BlackListStream:1.0.0</code>	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>		

- 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.
- 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.
- The `purchaseEvents.csv` contains credit card transactions data. Play it and send the transaction data.
- After sending sample events from `purchaseEvents.csv`, you will be able to see the outputs as follows.

```

Event: cardNo:4012-3253-3456-1881,
name:Srinath,
price:123.45
[2015-07-14 17:57:23,442] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger,
Event: cardNo:4012-3253-5632-6432,
name:Sachini,
price:123.56
[2015-07-14 17:57:24,442] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger,
Event: cardNo:4012-5678-2345-1234,
name:Mohan,
price:345.5
[2015-07-14 17:57:25,443] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: WhiteListPurchasesLogger,
Event: cardNo:4012-3253-5632-6432,
name:Sachini,
price:320.0

```

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 DAS 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.

```
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.

```
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.

```
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.

```
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 DAS 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 DAS 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 DAS from this file.

The screenshot shows the 'Event Stream Simulator' interface. At the top, there's a breadcrumb navigation: Home > Tools > Event Simulator. To the right is a 'Help' link. Below the navigation, the title 'Event Stream Simulator' is centered. There are two main sections: 'Send single event' and 'Send multiple events'.

Send single event: This section has a form with fields for 'Event Stream Name' (a dropdown menu labeled 'select event stream') and 'Stream Attributes'. At the bottom are 'Send' and 'Clear' buttons.

Send multiple events: This section has a table for inputting data from a file. The table has columns: File, Stream Configuration, Delay between events(ms), and Action. A row is shown with 'events.csv' selected in the 'File' column, 'org.wso2.event.sensor.stream:1.0.0' in 'Stream Configuration', '1000' in 'Delay between events(ms)', and three buttons in the 'Action' column: 'Play' (highlighted with a red box), 'Configure', and 'Delete'. Below the table, there are 'Choose File' and 'upload' buttons.

The events triggered in the DAS 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.

```
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:false,
meta_sensorId:100,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
triggered_time:1455520245294,
humidity:100.34,
sensorValue:23.4545
[2016-02-15 12:40:45,297] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: periodical,
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:true,
meta_sensorId:101,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
triggered_time:1455520245294,
humidity:100.34,
sensorValue:23.4545
[2016-02-15 12:40:45,297] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: periodical,
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:false,
meta_sensorId:103,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
triggered_time:1455520245294,
humidity:100.34,
sensorValue:23.4545
[2016-02-15 12:40:45,297] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: periodical,
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:true,
meta_sensorId:104,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
triggered_time:1455520245294,
humidity:100.34,
sensorValue:23.4545
[2016-02-15 12:40:50,004] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: cron,
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:false,
meta_sensorId:100,
meta_sensorName:temperature,
correlation_longitude:23.45656,
```

Sample 0120 - Using the Map Extension

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

Introduction

This sample demonstrates the usage of the 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`.

```
from dataIn
select name,map:createFromJSON(specificAttributesObj) as specAttrObjMap
insert into innerStream;
```

2. 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.

```
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.

```
from innerStreamTwo[not(temp is null)]
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 DAS server with the sample configuration numbered **0120**. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to `<DAS_HOME>/samples/cep/artifacts/0120` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`).

Executing the sample

Open a new tab in your CLI and execute the following ant command from the `<DAS_HOME>/samples/cep/producers/http` directory.

```
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 `<DAS_HOME>/samples/cep/producers/http/build.xml` file.

This builds the HTTP client and publishes the events in the `<DAS_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.

```
tishan@tishan-PC:~/packs/cep/wso2cep-4.2.0-SNAPSHOT/samples/producers$ ant -Durl=http://localhost:9763/endpoints/httpReceiver -Dsm=0120
Buildfile: /home/tishan/packs/cep/wso2cep-4.2.0-SNAPSHOT/samples/producers/http/build.xml

init:
[mkdir] Created dir: /home/tishan/packs/cep/wso2cep-4.2.0-SNAPSHOT/samples/producers/http/temp
[mkdir] Created dir: /home/tishan/packs/cep/wso2cep-4.2.0-SNAPSHOT/samples/producers/http/temp/classes

compile:
[javac] /home/tishan/packs/cep/wso2cep-4.2.0-SNAPSHOT/samples/producers/http/build.xml:71: warning: 'includeantruntime' was not set, defaulting to build.sysclasspath=last; set to false for repeatable builds
[javac] Compiling 2 source files to /home/tishan/packs/cep/wso2cep-4.2.0-SNAPSHOT/samples/producers/http/temp/classes
[javac] Note: /home/tishan/packs/cep/wso2cep-4.2.0-SNAPSHOT/samples/producers/http/src/main/java/org/wso2/carbon/sample/http/Http.java uses or overrides a deprecated API.
[javac] Note: Recompile with -Xlint:deprecation for details.
[copy] Copying 1 file to /home/tishan/packs/cep/wso2cep-4.2.0-SNAPSHOT/samples/producers/http/temp/classes

run:
[echo] Configure -Durl=xxxx and (-DfilePath=xxxx or -Dsm='sample number') optionally use -Dusername=xxxx -Dpassword=xxxx
[java] [main] INFO org.wso2.carbon.sample.http.Http - Starting WSO2 Http Client
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] {
[java]   "name": "Vehicle",
[java]   "numOfWheels": 5,
[java]   "specificAttributesObj": {
[java]     "temperature": "red",
[java]     "carColor": "#f00"
[java]   }
[java]
[java] [main] INFO org.wso2.carbon.sample.http.Http - Sending message:
[java] [main] INFO org.wso2.carbon.sample.http.Http -
[java] {
[java]   "name": "Car",
[java]   "numOfWheels": 4,
[java]   "specificAttributesObj": {
[java]     "temperature": "red",
[java]     "carColor": "#f00"
[java]   }
[java]

BUILD SUCCESSFUL
Total time: 2 seconds
```

JSON events are logged as shown below.

```
[2016-07-27 14:09:55,024] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: CHECK,
Event: name:Vehicle,
temperature:red
[2016-07-27 14:09:55,034] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: CHECK,
Event: name:Car,
temperature:red
```

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 DAS console.

The execution plan used in this sample is as follows.

- The following query defines the sensor input stream.

```
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.

```
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.

```
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.

```
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 DAS 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
 - overallMaxSensorReadingStreamPublisher

Executing the sample

Follow the steps below to execute the sample.

- Access the WSO2 DAS Management Console using the following URL, and log in using your credentials.
https://<DAS_HOST>:<DAS_PORT>/carbon/
- 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 DAS from this file.

The screenshot shows the WSO2 Data Analytics Server's Event Stream Simulator interface. It includes sections for sending single events and sending multiple events from a CSV file. The 'Send multiple events' section has a 'Play' button highlighted with a red circle.

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.

```
[2016-07-22 14:48:18,716] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: MaxSensorReadingPerRoom,
Event: sensorType:Light,
maxValue:44.12,
roomNo:3
[2016-07-22 14:48:18,716] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: MaxSensorReadingPerRoom
Event: sensorType:Humidity,
maxValue:69.73,
roomNo:4
[2016-07-22 14:48:18,717] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallMaxSensorReading
Event: sensorType:Pressure,
maxValue:94.41
[2016-07-22 14:48:18,717] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallMaxSensorReading,
Event: sensorType:Proximity,
maxValue:89.55
[2016-07-22 14:48:18,717] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallMaxSensorReading,
Event: sensorType:Light,
maxValue:96.11
[2016-07-22 14:48:18,717] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallMaxSensorReading,
Event: sensorType:Humidity,
maxValue:98.73
[2016-07-22 14:48:18,717] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallMaxSensorReading,
Event: sensorType:Sound,
maxValue:94.14
[2016-07-22 14:48:18,717] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallMaxSensorReading,
Event: sensorType:Temperature,
maxValue:82.08
[2016-07-22 14:48:18,718] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallAverageSensorReading,
Event: sensorType:Pressure,
avgValue:55.81000028337751
[2016-07-22 14:48:18,718] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallAverageSensorReading,
Event: sensorType:Proximity,
avgValue:58.60000099182129
[2016-07-22 14:48:18,718] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallAverageSensorReading,
Event: sensorType:Light,
avgValue:58.751666386922
[2016-07-22 14:48:18,718] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallAverageSensorReading,
Event: sensorType:Humidity,
avgValue:68.546000219726562
[2016-07-22 14:48:18,718] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallAverageSensorReading,
Event: sensorType:Sound,
avgValue:64.10571452549526
[2016-07-22 14:48:18,718] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: OverallAverageSensorReading,
Event: sensorType:Temperature,
avgValue:54.47666698031955
[2016-07-22 14:48:23,719] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: MaxSensorReadingPerRoom,
Event: sensorType:Humidity,
maxValue:57.23,
roomNo:7
[2016-07-22 14:48:23,721] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: MaxSensorReadingPerRoom,
Event: sensorType:Light,
maxValue:73.02,
roomNo:10
```

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 the DAS Event Simulator to input data and the logger publisher for logging the outputs to the DAS 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 DAS 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 `<DAS_HOME>/samples/cep/artifacts/0301` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`).

Executing the sample

1. Log into the DAS Management Console that is located at <https://localhost:9443/carbon>.
2. In the **Tools** tab, click **Event Simulator** to open the **Event Stream Simulator** page. In the **Send multiple events** section, the `intruder.csv` file is displayed. This file contains some sample data. Click **Play** to start sending sample events from the file.

The values of the `IntruderOutputStream` are logged in the WSO2 DAS console as shown below.

```
[2015-08-12 13:58:52,034] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: ***** intruder detection results *****,
Event: root_shell:1.0,
su_attempted:148.0,
num_root:71.0,
num_file_creations:35.0,
num_shells:0.0,
num_access_files:33.6,
num_outbound_cmds:0.627,
is_host_login:50.0,
is_guest_login:1.0,
count:2.0,
srv_count:3.0,
serror_rate:4.0,
srv_serror_rate:5.0,
Predicted_response:1,
Probability_0:,1,
Probability_1:0.018258426966292134
```

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.

```
@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.

```
<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>
```

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.

```

<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.

```
run: [echo] Configure -DstreamId=xxxx:x.x.x and ( -Dsn='sample number' or -DfilePath=xxxx ) optionally use -Dprotocol='thrift/binary' -Dhost=xxxx -Dport=xxxx -Dusername=xxxx -Dpassword=xxxx -Devents=xx -Ddelay='delay between events in ms'
[java] [thrift localhost, 7611, admin, admin, analytics_Statistics:1.3.0, 2001, , 10, 1000]
[java] Starting WSO2 Event Client
[java] [main] INFO org.wso2.carbon.sample.wso2event.Client - StreamDefinition used :
[java]   "name": "analytics_Statistics",
[java]   "version": "1.3.0",
[java]   "nickName": "",
[java]   "description": "",
[java]   "metadata": [
[java]     {
[java]       "name": "ipAdd",
[java]       "type": "STRING"
[java]     },
[java]     {
[java]       "name": "index",
[java]       "type": "LONG"
[java]     },
[java]     {
[java]       "name": "timestamp",
[java]       "type": "LONG"
[java]     },
[java]     {
[java]       "name": "nanoTime",
[java]       "type": "LONG"
[java]     }
[java]   ],
[java]   "payloadData": [
[java]     {
[java]       "name": "userID",
[java]       "type": "STRING"
[java]     },
[java]     {
[java]       "name": "searchTerms",
[java]       "type": "STRING"
[java]     }
[java]   ]
[java] }
```

You view the output events received by the consumer from WSO2 CEP via the terminal opened in step 2 above as shown below.

```
[java] ,
[java] Event{
[java]   streamId='filteredStats:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[203.93.106.23, 2, 1412271770130, 1412271770130983000],
[java]   correlationData=null,
[java]   payloadData=[chris@org1.com],
[java]   arbitraryDataMap=null,
[java] }
[java] ,
[java] Event{
[java]   streamId='filteredStats:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[203.94.106.11, 4, 1412271770131, 1412271770131016000],
[java]   correlationData=null,
[java]   payloadData=[sam@org2.com],
[java]   arbitraryDataMap=null,
[java] }
[java] ,
[java] Event{
[java]   streamId='filteredStats:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[192.168.1.23, 9, 1412271770131, 1412271770131153000],
[java]   correlationData=null,
[java]   payloadData=[lang@ws2.org],
[java]   arbitraryDataMap=null,
[java] }
[java] ,
[java] 
```

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 ipCount, sum(frequency) as totalCount
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.
 - a. Do the following changes in the `<CEP_HOME>/repository/conf/axis2/axis2.xml` file, to enable clustering.

```

<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>

```

- b. 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.FileSystem
PersistenceStore">
                <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: `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.

```
[2016-02-16 10:58:31,199] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 1 totalCount = 2
[2016-02-16 10:58:31,210] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 2 totalCount = 7
[2016-02-16 10:58:31,213] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 3 totalCount = 8
[2016-02-16 10:58:31,217] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 4 totalCount = 10
[2016-02-16 10:58:31,220] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 5 totalCount = 13
[2016-02-16 10:58:31,223] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 6 totalCount = 19
```

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.

```
[2016-02-16 10:59:14,563] INFO {org.wso2.siddhi.core.util.snapshot.SnapshotService} - Taking snapshot ...
[2016-02-16 10:59:14,564] INFO {org.wso2.siddhi.core.util.snapshot.SnapshotService} - Taking snapshot finished.
[2016-02-16 10:59:14,564] INFO {org.wso2.siddhi.core.util.snapshot.SnapshotService} - Snapshot serialization started ...
[2016-02-16 10:59:14,569] INFO {org.wso2.siddhi.core.util.snapshot.SnapshotService} - Snapshot serialization finished.
```

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.

```
[2016-02-16 11:00:07,603] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 7 totalCount = 21
[2016-02-16 11:00:07,616] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 8 totalCount = 26
[2016-02-16 11:00:07,618] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 9 totalCount = 27
[2016-02-16 11:00:07,621] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 10 totalCount = 29
[2016-02-16 11:00:07,624] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 11 totalCount = 32
[2016-02-16 11:00:07,627] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: login information, Event: ipCount = 12 totalCount = 38
```

Event processing continues from the previous state even after you restart the server. The ipCount of the first event is displayed as 7 because you already sent 6 events in step 1. The totalCount is also calculated by considering the previous events as well.

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 DAS in High Availability (HA) mode. This uses the following window query to demonstrate how a processing continues even after one server shuts down.

```
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 DAS 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](#). Summarized steps are as follows.
 - a. 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.
 - b. In the <CEP_HOME>/repository/conf/axis2/axis2.xml file, enable clustering in **wka** mode on both nodes.
 - c. 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.
 - d. 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.

```
[2015-05-21 17:51:38,456] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:1, sumPrice:500.75
[2015-05-21 17:51:38,460] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:2, sumPrice:2000.75
[2015-05-21 17:51:38,463] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:3, sumPrice:2451.25
```

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 **Node1** (active node) receiving the output events in the logs of it in the CLI as shown below.

```
[2015-05-21 17:52:13,805] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:4, sumPrice:2952.0
[2015-05-21 17:52:13,809] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:5, sumPrice:4452.0
[2015-05-21 17:52:13,811] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:6, sumPrice:4902.5
```

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.

```
[2015-05-21 17:54:01,837] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:7, sumPrice:5403.25
[2015-05-21 17:54:01,846] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:8, sumPrice:6903.25
[2015-05-21 17:54:01,848] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: pizza order count, Event: totalOrders:9
```

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 DAS with Apache Storm in the distributed mode, and run the sample query below in a local/distributed Storm cluster.

```
@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 the analyticsStats stream, and inserts the results into the stream named processedStream.

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 <DAS_HOME>/repository/conf/axis2/axis2.xml file, to enable Hazelcast clustering in WSO2 CEP.

```
<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>
```

4. Do the following changes in the <DAS_HOME>/repository/conf/event-processor.xml file to disable the HA processing mode, enable the Distributed processing mode, and configure WSO2 DAS to run with Apache Storm.

```

<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 DAS 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 <DAS_HOME>/repository/conf/cep/storm/storm.yaml file on all nodes, for WSO2 DAS to communicate with Apache Storm. However, these configurations are not necessary for this sample, if you run Apache Storm locally.

6. 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 DAS 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`.

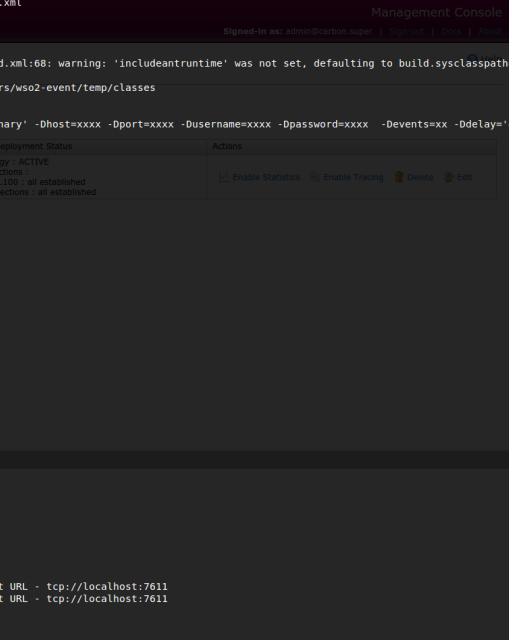
Executing the sample

Follow the steps below to execute the sample.

1. Navigate to the `<Das_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`

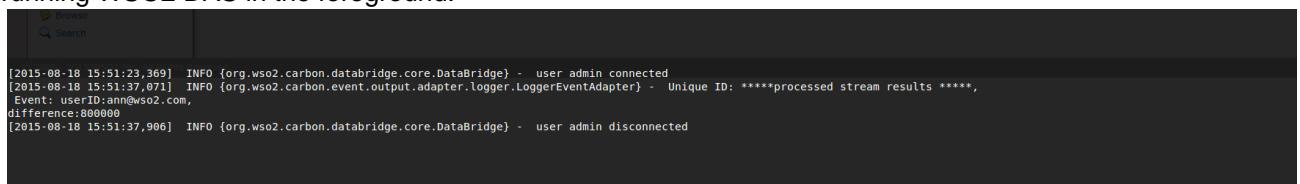
The other optional parameters that can be used in the above command are defined in the `<Das_Home>/cep/samples/producers/wso2-event/build.xml` file.

2. This builds and runs the `wso2event` producer that will send analytics statistics data to the DAS server. You view the details of the events that are sent as shown below.



```
Buildfile: /home/lasantha/WSO2-Staging/cep400/local-build/17-Aug-build/wso2cep-4.0.0-SNAPSHOT/samples/producers/wso2-event/build.xml
init:
compile:
[javac] /home/lasantha/WSO2-Staging/cep400/local-build/17-Aug-build/wso2cep-4.0.0-SNAPSHOT/samples/producers/wso2-event/build.xml:68: warning: 'includeantruntime' was not set, defaulting to build.sysclasspath
last; set to false for repeatable builds
[copy] Copying 1 file to /home/lasantha/WSO2-Staging/cep400/local-build/17-Aug-build/wso2cep-4.0.0-SNAPSHOT/samples/producers/wso2-event/temp/classes
run:
[echo] Configure -StreamId=xxxx.x.x and (-Dsn='sample number' or -DfilePath=xxxx ) optionally use -Dprotocol='thrift/binary' -Dhost=xxxx -Dport=xxxx -Dusername=xxxx -Dpassword=xxxx -Devents=xx -Ddelay='
[elide between events in ms'
[java] [thrift, localhost, 7611, admin, admin, analytics_Statistics:1.3.0, 0504, , 100, 1000]
[java] Starting WSO2 Event Client
[java] [main] INFO org.wso2.carbon.sample.wso2event.Client - StreamDefinition used :{
[java]   "name": "analytics_Statistics",
[java]   "version": "1.3.0",
[java]   "nickName": "",
[java]   "description": "",
[java]   "metadata": [
[java]     {
[java]       "name": "ipAdd",
[java]       "type": "STRING"
[java]     },
[java]     "lat",
[java]     "long",
[java]     "name": "index",
[java]     "type": "LONG"
[java]   ],
[java]   "events": [
[java]     {
[java]       "name": "timestamp",
[java]       "type": "LONG"
[java]     },
[java]     "now",
[java]     {
[java]       "name": "nanoTime",
[java]       "type": "LONG"
[java]     }
[java]   ],
[java]   "payloadData": [
[java]     {
[java]       "name": "userID",
[java]       "type": "STRING"
[java]     },
[java]     {
[java]       "name": "searchTerms",
[java]       "type": "STRING"
[java]     }
[java]   ]
[java] }
[java] [main] INFO org.wso2.carbon.databridge.agent.endpoint.DataEndpoint - Shutdown triggered for data publisher endpoint URL - tcp://localhost:7611
[java] [main] INFO org.wso2.carbon.databridge.agent.endpoint.DataEndpoint - Completed shutdown for data publisher endpoint URL - tcp://localhost:7611
BUILD SUCCESSFUL
Total time: 16 seconds
```

The output events received by the consumer from WSO2 DAS can be viewed via the terminal opened when running WSO2 DAS in the foreground.



```
[2015-08-18 15:51:23,369] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
[2015-08-18 15:51:37,071] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: *****processed stream results *****,
Event: userID:ann@wso2.com,
difference:800000
[2015-08-18 15:51:37,906] INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin disconnected
```

Sample 1001 - WSO2 CEP Geo Dashboard

Introduction

This sample demonstrates how the geo dashboard of the WSO2 DAS can be used. Full details about setting up, configuring and running this sample is available at [Geo Dashboard](#).

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 DAS server.
 - On Windows: `<DAS_HOME>\bin\wso2server.bat --run`
 - On Linux/Solaris/Mac OS: `sh <DAS_HOME>/bin/wso2server.sh`
 2. Copy the contents in the `<DAS_HOME>/samples/cep/artifacts/1501/carbonapps` directory to the `<DAS_HOME>/repository/deployment/server/carbonapps` directory.
- If the `carbonapps` directory does not already exist in the `<DAS_HOME>/repository/deployment/server/carbonapps` directory, create it and copy the required content to it.
3. Copy the contents in the `<DAS_HOME>/samples/cep/artifacts/1501/eventsimulatorfiles` directory to the `<DAS_HOME>/repository/deployment/server/eventsimulatorfiles` directory.
 4. Log into the WSO2 DAS Management Console via the following URL.
`https://<DAS_HOST>:<DAS_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.

Home > Tools > Event Simulator

Event Stream Simulator

Send single event

Event Stream Name *	<input type="text" value="realTimeStreamData:1.0.0"/> ✓ select event stream
Stream Attributes	
<input type="button" value="Send"/> <input type="button" value="Clear"/>	

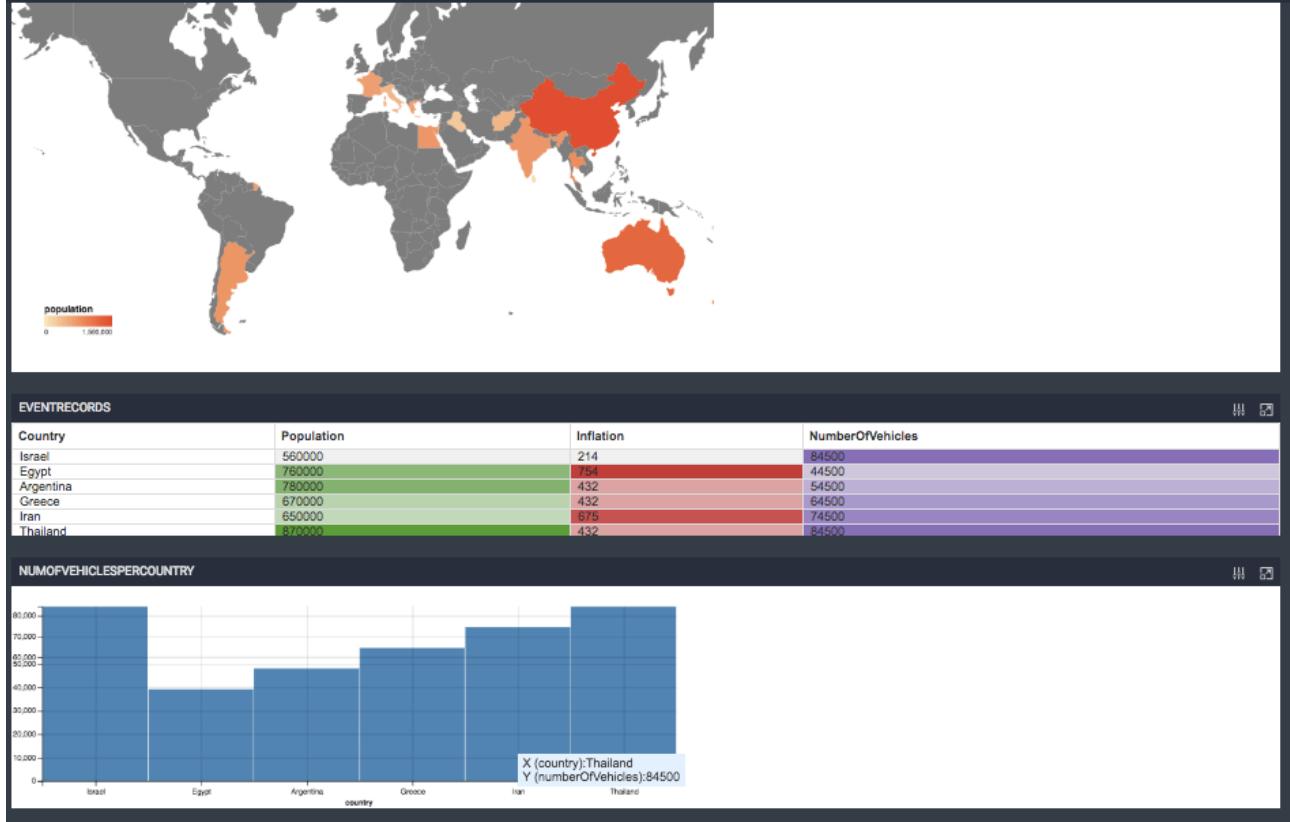
Send multiple events

Input Data by File [switch to configure database for simulation](#)

File	Stream Configuration	Delay between events(ms)	Action
events.csv	click the configure button	click the configure button	<input type="button" value="Configure"/> <input type="button" value="Delete"/>
realTimeDashboardData.csv	realTimeStreamData:1.0.0	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="Choose File"/> No file chosen		<input type="button" value="upload"/>	

Rights Reserved.

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 XML, 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
- Prerequisites
- Building the sample
- Executing the sample

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 DAS 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.
- Loads the events stored in the <DAS_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 into the DAS 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 <DAS_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.

[Send multiple events](#)

Input Data by File switch to configure database for simulation		Action
File	Stream Configuration	
events.csv	org.wso2.event.sensor.stream:1.0.0	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
Choose File No file chosen	<input type="button" value="upload"/>	

Send multiple events

Input Data by File		Stream Configuration	Delay between events(ms)	Action
File	events.csv	org.wso2.event.sensor.stream:1.0.0	1000	<input type="button" value="Pause"/> <input type="button" value="Stop"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="Browse..."/> No file selected.		<input type="button" value="upload"/>		

The JSON events that are published to the DAS server are logged in the CLI as shown below.

```
[2015-06-24 17:58:39,789] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: JSONLogger, Event: {"event":{"metaData":{"timestamp":"199008131245","isPowerSaverEnabled":"false","sensorId":"100","sensorName":"temperature","correlationData":{"longitude":"23.45656","latitude":"7.12324"},"payloadData":{"humidity":100.34,"sensorValue":23.4545}}}}
[2015-06-24 17:58:39,790] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: JSONLogger, Event: {"event":{"metaData":{"timestamp":"199008131245","isPowerSaverEnabled":"true","sensorId":101,"sensorName":temperature,"correlationData":{"longitude":23.45656,"latitude":7.12324,"payloadData":{"humidity":100.34,"sensorValue":23.4545}}}}
[2015-06-24 17:58:39,791] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: JSONLogger, Event: {"event":{"metaData":{"timestamp":"199008131245","isPowerSaverEnabled":"false","sensorId":103,"sensorName":temperature,"correlationData":{"longitude":23.45656,"latitude":7.12324,"payloadData":{"humidity":100.34,"sensorValue":23.4545}}}}
[2015-06-24 17:58:39,791] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: JSONLogger, Event: {"event":{"metaData":{"timestamp":"199008131245","isPowerSaverEnabled":true,"sensorId":104,"sensorName":temperature,"correlationData":{"longitude":23.45656,"latitude":7.12324,"payloadData":{"humidity":100.34,"sensorValue":23.4545}}}}
```

Sample 0052 - Publishing Custom JSON Events via Logger Transport

- [Introduction](#)
- [Prerequisites](#)
- [Building the sample](#)
- [Executing the sample](#)

Introduction

This sample demonstrates how to publish custom JSON events via the 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 DAS 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`.
- Loads the events stored in the `<DAS_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 into the DAS 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 `<DAS_HOME>/samples/cep/artifacts/0052/eventsimulatorfiles/events.csv` file to the publisher.

Send multiple events

Input Data by File		Stream Configuration	Action	
File	events.csv	org.wso2.event.sensor.stream:1.0.0	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>	
<input type="button" value="Choose File"/> No file chosen		<input type="button" value="upload"/>		

The JSON events that are published to the DAS server are logged in the CLI as shown below.

```

Event: {
    "Sensor Data": {
        "equipment related data": {
            "timestamp": 199008131245,
            "isPowerSaverEnabled": true,
            "sensorId": 101,
            "sensorName": "temperature"
        },
        "location data": {
            "longitude": 23.45656,
            "latitude": 7.12324
        },
        "sensor data": {
            "humidity": 100.34,
            "sensorValue": 23.4545
        }
    }
}
[2016-07-12 14:11:05,567] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: loggerWithJSONMapping,
Event: {
    "Sensor Data": {
        "equipment related data": {
            "timestamp": 199008131245,
            "isPowerSaverEnabled": false,
            "sensorId": 103,
            "sensorName": "temperature"
        },
        "location data": {
            "longitude": 23.45656,
            "latitude": 7.12324
        },
        "sensor data": {
            "humidity": 100.34,
            "sensorValue": 23.4545
        }
    }
}
[2016-07-12 14:11:06,568] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: loggerWithJSONMapping,
Event: {
    "Sensor Data": {
        "equipment related data": {
            "timestamp": 199008131245,
            "isPowerSaverEnabled": true,
            "sensorId": 104,
            "sensorName": "temperature"
        },
        "location data": {
            "longitude": 23.45656,
            "latitude": 7.12324
        },
        "sensor data": {
            "humidity": 100.34,
            "sensorValue": 23.4545
        }
    }
}
[2016-07-12 14:11:19,594] INFO {org.wso2.carbon.event.processor.core.internal.CarbonEventManagementService} - Starting polling event receivers

```

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 the logger transport. This sample does not do any processing on the outgoing events. 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 DAS 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`.
- Loads the events stored in the `<DAS_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 into the DAS 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 <DAS_HOME> / samples/cep/artifacts/0053/eventsimulatorfiles/events.csv file to the publisher.

[Send multiple events](#)

File	Stream Configuration	Action
events.csv	org.wso2.event.sensor.stream:1.0.0	Play Configure Delete

Choose File No file chosen upload

The XML events that are published to the DAS server are logged in the CLI as shown below.

```
[2015-06-24 18:21:27,441] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: xmlLogger, Event: <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>false</isPowerSaverEnabled><sensorId>100</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
[2015-06-24 18:21:27,443] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: xmlLogger, Event: <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>true</isPowerSaverEnabled><sensorId>101</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
[2015-06-24 18:21:27,444] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: xmlLogger, Event: <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>false</isPowerSaverEnabled><sensorId>103</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
[2015-06-24 18:21:27,445] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: xmlLogger, Event: <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>true</isPowerSaverEnabled><sensorId>104</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
```

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 DAS 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 <DAS_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 into the DAS 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 <DAS_HOME>/samples/cep/artifacts/0054/eventsimulatorfiles/events.csv file to the publisher.

Send multiple events

Input Data by File 		Action
File	Stream Configuration	
events.csv	org.wso2.event.sensor.stream:1.0.0	  
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>	

The XML events that are published to the DAS server are logged in the CLI as shown below.

```
[2015-06-24 18:30:21,233] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: xmlMappingLogger, Event: <SensorData>
    <equipmentRelatedData>
        <timestamp>199008131245</timestamp>
        <isPowerSaverEnabled>false</isPowerSaverEnabled>
        <sensorId>100</sensorId>
        <sensorName>temperature</sensorName>
    </equipmentRelatedData>
    <locationData>
        <longitude>23.45656</longitude>
        <latitude>7.12324</latitude>
    </locationData>
    <sensorData>
        <humidity>100.34</humidity>
        <sensorValue>23.4545</sensorValue>
    </sensorData>
</SensorData>
[2015-06-24 18:30:21,234] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: xmlMappingLogger, Event: <SensorData>
    <equipmentRelatedData>
        <timestamp>199008131245</timestamp>
        <isPowerSaverEnabled>true</isPowerSaverEnabled>
        <sensorId>101</sensorId>
        <sensorName>temperature</sensorName>
    </equipmentRelatedData>
    <locationData>
        <longitude>23.45656</longitude>
        <latitude>7.12324</latitude>
    </locationData>
    <sensorData>
        <humidity>100.34</humidity>
        <sensorValue>23.4545</sensorValue>
    </sensorData>
</SensorData>
[2015-06-24 18:30:21,235] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: xmlMappingLogger, Event: <SensorData>
    <equipmentRelatedData>
        <timestamp>199008131245</timestamp>
        <isPowerSaverEnabled>false</isPowerSaverEnabled>
        <sensorId>103</sensorId>
        <sensorName>temperature</sensorName>
    </equipmentRelatedData>
    <locationData>
        <longitude>23.45656</longitude>
        <latitude>7.12324</latitude>
    </locationData>
    <sensorData>
        <humidity>100.34</humidity>
        <sensorValue>23.4545</sensorValue>
    </sensorData>
</SensorData>
[2015-06-24 18:30:21,236] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: xmlMappingLogger, Event: <SensorData>
    <equipmentRelatedData>
        <timestamp>199008131245</timestamp>
        <isPowerSaverEnabled>true</isPowerSaverEnabled>
        <sensorId>104</sensorId>
        <sensorName>temperature</sensorName>
    </equipmentRelatedData>
    <locationData>
        <longitude>23.45656</longitude>
        <latitude>7.12324</latitude>
    </locationData>
```

Sample 0055 - Publishing Text Events via Logger Transport

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

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 DAS 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 `<DAS_HOME>/samples/cep/artifacts/0055/eventsimulatorfiles/events.csv` file to the Event Simulator.

Executing the sample

Follow the steps below to execute the sample.

- Log into the DAS Management Console.
- Click **Tools**, and then click **Event Simulator**.
- Click **Play** on the corresponding event stream as shown below, to send the events in the `<DAS_HOME>/samples/cep/artifacts/0055/eventsimulatorfiles/events.csv` file to the publisher.

Send multiple events

Input Data by File		Action
File	Stream Configuration	
events.csv	<code>org.wso2.event.sensor.stream:1.0.0</code>	Play Configure Delete
Choose File No file chosen	upload	

The text events that are published to the DAS server are logged in the CLI as shown below.

```
[2015-06-24 18:38:02,278] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:false,
meta_sensorId:100,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
humidity:100.34,
sensorValue:23.4545
[2015-06-24 18:38:02,278] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:true,
meta_sensorId:101,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
humidity:100.34,
sensorValue:23.4545
[2015-06-24 18:38:02,279] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:false,
meta_sensorId:103,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
humidity:100.34,
sensorValue:23.4545
[2015-06-24 18:38:02,279] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: org.wso2.event.statistics.logger,
Event: meta_timestamp:199008131245,
meta_isPowerSaverEnabled:true,
meta_sensorId:104,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
humidity:100.34,
sensorValue:23.4545
```

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 DAS 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`.
- Loads the events stored in the `<Das_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 into the DAS 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 `<Das_Home>/samples/cep/artifacts/0056/eventsimulatorfiles/events.csv` file to the publisher.

Send multiple events

Input Data by File 		Action
File	Stream Configuration	
events.csv	<code>org.wso2.event.sensor.stream:1.0.0</code>	Play Configure Delete
Choose File No file chosen	upload	

The test events that are published to the DAS server are logged in the CLI as shown below.

```
[2015-06-24 18:44:21,095] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: logger for, Event: Sensor Data Information

temperature Sensor related data.
- sensor id: 100
- time-stamp: 199008131245
- power saving enabled: false
Location
- longitude: 23.45656
- latitude: 7.12324
Values
- temperature: 23.4545
- humidity: 100.34

[2015-06-24 18:44:21,096] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: logger for, Event: Sensor Data Information

temperature Sensor related data.
- sensor id: 101
- time-stamp: 199008131245
- power saving enabled: true
Location
- longitude: 23.45656
- latitude: 7.12324
Values
- temperature: 23.4545
- humidity: 100.34

[2015-06-24 18:44:21,096] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: logger for, Event: Sensor Data Information

temperature Sensor related data.
- sensor id: 103
- time-stamp: 199008131245
- power saving enabled: false
Location
- longitude: 23.45656
- latitude: 7.12324
Values
- temperature: 23.4545
- humidity: 100.34

[2015-06-24 18:44:21,096] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: logger for, Event: Sensor Data Information

temperature Sensor related data.
- sensor id: 104
- time-stamp: 199008131245
- power saving enabled: true
Location
- longitude: 23.45656
- latitude: 7.12324
Values
- temperature: 23.4545
- humidity: 100.34
```

Sample 0057 - Publishing WSO2 Events via WSO2Event Transport

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

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 events. 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 DAS 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`.
- Loads the events that are stored in the `events.csv` file via the Event Simulator.

Executing the sample

Navigate to the `<DAS_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 `<DAS_HOME>/samples/cep/consumers/wso2-event/build.xml` file.

This builds a client that fetches the published events in the `<DAS_HOME>/samples/cep/artifacts/0057/eventsimulatorfiles/events.csv` file from the `emailPublisher` endpoint.

Go to the Event Simulator and click **Play** to send events stored in the `events.csv` file.

Home > Tools > Event Simulator [Help](#)

Event Stream Simulator

Send multiple events

Input Data by File [switch to add configuration for simulate by database](#)

File	Stream Configuration	Action
events.csv	org.wso2.event.sensor.stream:1.0.0	Play Configure Delete

[Browse...](#) No file selected. [upload](#)

Received events are logged in the Wso2Event client terminal console as shown below.

```
[java] eventListSize=1 eventList [
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131245, false, 100, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
[java] ] for username admin
[java] eventListSize=2 eventList [
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131285, true, 101, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
[java] ,
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131325, true, 102, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
[java] ] for username admin
[java] eventListSize=1 eventList [
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131345, false, 103, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
[java] ] for username admin
```

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 DAS 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 `<DAS_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 `<DAS_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 `<DAS_HOME>/samples/cep/consumers/wso2-event/build.xml` file.

This builds the WSO2Event client that fetches the published events in the `<DAS_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv` file from the `eventPublisher` endpoint as shown below.

```
run:
[echo] To configure host, port and events use -Dhost=xxxx -Dport=xxx -Dprotocol=thrift/binary -Dsn=sampleNumber
[java] StreamDefinition of 'org.wso2.event.sensor.stream.map:1.0.0' added to store
[java] StreamDefinition of 'org.wso2.event.sensor.stream:1.0.0' added to store
[java] Thrift Server started at 0.0.0.0
[java] Thrift SSL port : 7761
[java] Thrift port : 7661
[java] Test Server Started
```

2. Log into the DAS 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 `<DAS_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv` file to the publisher.

Send multiple events

Input Data by File		Action
File	Stream Configuration	Play <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="file" value="events.csv"/> <input type="button" value="Choose File"/>	org.wso2.event.sensor.stream:1.0.0	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="upload"/>		

The WSO2 events that are published to the DAS server are logged in the consumer terminal as shown below.

```
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream.map:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131345, false, 103, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
[java] ] for username admin
[java] eventListSize=3 eventList [
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream.map:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131245, false, 100, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
[java] ,
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream.map:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131285, true, 101, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
[java] ,
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream.map:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131325, true, 102, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
[java] ] for username admin
[java] eventListSize=1 eventList [
[java] Event{
[java]   streamId='org.wso2.event.sensor.stream.map:1.0.0',
[java]   timeStamp=0,
[java]   metaData=[199008131345, false, 103, temperature],
[java]   correlationData=[23.45656, 7.12324],
[java]   payloadData=[100.34, 23.4545],
[java]   arbitraryDataMap=null,
[java] }
```

Sample 0059 - Publishing Map and Text Events via JMS Transport - ActiveMQ

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

Introduction

This sample demonstrates how to publish Map and Text events via JMS transport. You consume the published events in WSO2 DAS 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.

```
./activemq console
```

3. Configure WSO2 DAS by adding relevant libraries to [support JMS transport](#).

4. Configure the sample client by adding relevant jars. For more information, see [setting up JMS for JMS sample clients](#).

5. Open the <DAS_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?broker
topic.topicMap = topicMap
topic.topicText = topicText
```

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/samples/cep/artifacts/0059/eventsimulatorfiles/events.csv file to the Event Simulator.

Executing the sample

1. Open another terminal and navigate to the <DAS_HOME>/samples/cep/consumers/jms directory.

and run one of the 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 sends an event flow through the stream to the JMS publisher.

The events that are published by DAS are logged in the JMS consumer console as follows.

Map formatted events:

```
[java] Received Map Message : {isPowerSaverEnabled=false, timestamp=199008131245, sensorValue=23.4545, humidity=100.34, sensorId=100, longitude=23.45656, latitude=7.12324, sensorName=temperature}
[java] Received Map Message : {isPowerSaverEnabled=true, timestamp=199008131245, sensorValue=23.4545, humidity=100.34, sensorId=101, longitude=23.45656, latitude=7.12324, sensorName=temperature}
[java] Received Map Message : {isPowerSaverEnabled=false, timestamp=199008131245, sensorValue=23.4545, humidity=100.34, sensorId=103, longitude=23.45656, latitude=7.12324, sensorName=temperature}
[java] Received Map Message : {isPowerSaverEnabled=true, timestamp=199008131245, sensorValue=23.4545, humidity=100.34, sensorId=104, longitude=23.45656, latitude=7.12324, sensorName=temperature}
```

for Text formatted events:

```
[java] Listening for messages
[java] Received Text Message : meta_timestamp:199008131245,
[java] meta_isPowerSaverEnabled:false,
[java] meta_sensorId:100,
[java] meta_sensorName:temperature,
[java] correlation_longitude:23.45656,
[java] correlation_latitude:7.12324,
[java] humidity:100.34,
[java] sensorValue:23.4545
[java] Received Text Message : meta_timestamp:199008131245,
[java] meta_isPowerSaverEnabled:true,
[java] meta_sensorId:101,
[java] meta_sensorName:temperature,
[java] correlation_longitude:23.45656,
[java] correlation_latitude:7.12324,
[java] humidity:100.34,
[java] sensorValue:23.4545
[java] Received Text Message : meta_timestamp:199008131245,
[java] meta_isPowerSaverEnabled:false,
[java] meta_sensorId:103,
[java] meta_sensorName:temperature,
[java] correlation_longitude:23.45656,
[java] correlation_latitude:7.12324,
[java] humidity:100.34,
[java] sensorValue:23.4545
[java] Received Text Message : meta_timestamp:199008131245,
[java] meta_isPowerSaverEnabled:true,
[java] meta_sensorId:104,
[java] meta_sensorName:temperature,
[java] correlation_longitude:23.45656,
[java] correlation_latitude:7.12324,
[java] humidity:100.34,
[java] sensorValue:23.4545
```

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 DAS using a JMS client that is subscribed to the 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 DAS by adding relevant libraries to support JMS transport.
3. Configure a sample client by adding relevant jars. See [setting up JMS for JMS sample clients](#).
4. Open the <DAS_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@clientid/default?br
topic.topicMap = topicMap
topic.topicJSON = topicJSON
```

- Start Qpid Broker with **./qpid-server start** command by navigating to <QPID-BROKER_HOME>/bin.

Building the sample

Start the WSO2 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_HOME>/sample/cep/artifacts/0060.
- Creates a stream with ID org.wso2.event.sensor.stream:1.0.0.
- Creates event publishers named jmsPublisherMap and jmsPublisherText.
- Hot deploys the event.csv file along with the event configuration xml.

Executing the sample

- Start WSO2 DAS with sample 0060 configurations. For more information, see [Starting sample CEP configurations](#).
- Open another terminal and navigate to the <DAS_HOME>/samples/cep/consumers/jms directory.

Run one of the following commands according to the type of the topic name of the messages to be consumed.

```
ant -Dbroker=qpid -DtopicName=topicMap
ant -Dbroker=qpid -DtopicName=topicJSON
```

It subscribes to the Qpid broker under the mentioned topic name.

- Play the event.csv file deployed using the Event Simulator. This sends an event flow through the stream to the JMS publishers.
- The events that are published by WSO2 DAS are logged in the JMS consumer console as follows.

Map formatted events:

```
[java] Received Map Message : {timestamp=199008131245, humidity=100.34, sensorId=100, longitude=23.45656, powerSaving=false, latitude=7.12324, sensorName=temperature, temperature=23.4545}
[java] Received Map Message : {timestamp=199008131245, humidity=100.34, sensorId=101, longitude=23.45656, powerSaving=true, latitude=7.12324, sensorName=temperature, temperature=23.4545}
[java] Received Map Message : {timestamp=199008131245, humidity=100.34, sensorId=103, longitude=23.45656, powerSaving=false, latitude=7.12324, sensorName=temperature, temperature=23.4545}
[java] Received Map Message : {timestamp=199008131245, humidity=100.34, sensorId=104, longitude=23.45656, powerSaving=true, latitude=7.12324, sensorName=temperature, temperature=23.4545}
```

Text formatted events:

```
[java] Received Text Message : {"Sensor Data":{"equipment related data":{"timestamp":199008131245, "isPowerSaverEnabled":false,"sensorId":100,"sensorName":"temperature"}, "location data":{"longitude":23.45656, "latitude":7.12324}, "sensor data":{"humidity":100.34,"sensorValue":23.4545}}
[java] Received Text Message : {"Sensor Data":{"equipment related data":{"timestamp":199008131245, "isPowerSaverEnabled":true,"sensorId":101,"sensorName":"temperature"}, "location data":{"longitude":23.45656, "latitude":7.12324}, "sensor data":{"humidity":100.34,"sensorValue":23.4545}}
[java] Received Text Message : {"Sensor Data":{"equipment related data":{"timestamp":199008131245, "isPowerSaverEnabled":false,"sensorId":103,"sensorName":"temperature"}, "location data":{"longitude":23.45656, "latitude":7.12324}, "sensor data":{"humidity":100.34,"sensorValue":23.4545}}
[java] Received Text Message : {"Sensor Data":{"equipment related data":{"timestamp":199008131245, "isPowerSaverEnabled":true,"sensorId":104,"sensorName":"temperature"}, "location data":{"longitude":23.45656, "latitude":7.12324}, "sensor data":{"humidity":100.34,"sensorValue":23.4545}}
```

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 DAS, start WSO2 MB (version 3.0.0 or later).
2. Configure WSO2 DAS by adding relevant libraries to [support JMS transport](#).
3. Configure a sample client by adding relevant jars. See [setting up JMS for JMS sample clients](#).
4. Open the `<DAS_HOME>/repository/conf/jndi.properties` file and register a connection factory named `TopicConnectionFactory` by entering the following in the `register some connection factories` section. Then add the topics to be sent to the Qpid broker.

```
connectionfactory.TopicConnectionFactory=amqp: //admin:admin@clientid/carbon?bro
topic.topicMap = topicMap
topic.topicXML = topicXML
```

Building the sample

Start the WSO2 DAS server with the sample configuration numbered 0061 with a port offset because WSO2 MB is running 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 `<DAS_HOME>/repository/deployment/server` to `<DAS_HOME>/sample/cep/artifacts/0061`.
- Creates a stream with ID `org.wso2.event.sensor.stream:1.0.0`.
- Creates event publishers named `jmsPublisherMap` and `jmsPublisherXML`.
- Hot deploys the `event.csv` file along with the `event configuration.xml` file.

Executing the sample

1. Start WSO2 DAS with sample **0061** configurations. For more information, see [Starting sample CEP configurations](#).
2. Issue the following commands from the `<DAS_HOME>/samples/cep/consumers/jms` directory using a separate terminal for each command. The commands should be issued according to the type of the topic name of the messages to be consumed.

```
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. For detailed instructions to play a file, see [Sending Multiple Events Using a CSV File](#).
4. Events published by WSO2 DAS are logged in the JMS consumer console as shown below.

for Map formatted events:

```
[java] Received Map Message : {isPowerSaverEnabled=false, timestamp=199008131245, sensorValue=23.4545, humidity=100.34, sensorId=100, longitude=23.45656, latitude=7.12324, sensorName=temperature}
[java] Received Map Message : {isPowerSaverEnabled=true, timestamp=199008131245, sensorValue=23.4545, humidity=100.34, sensorId=101, longitude=23.45656, latitude=7.12324, sensorName=temperature}
[java] Received Map Message : {isPowerSaverEnabled=false, timestamp=199008131245, sensorValue=23.4545, humidity=100.34, sensorId=103, longitude=23.45656, latitude=7.12324, sensorName=temperature}
[java] Received Map Message : {isPowerSaverEnabled=true, timestamp=199008131245, sensorValue=23.4545, humidity=100.34, sensorId=104, longitude=23.45656, latitude=7.12324, sensorName=temperature}
```

for XML formatted events:

```
[java] Received Text Message : <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>false</isPowerSaverEnabled><sensorId>100</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
[java] Received Text Message : <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>true</isPowerSaverEnabled><sensorId>101</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
[java] Received Text Message : <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>false</isPowerSaverEnabled><sensorId>103</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
[java] Received Text Message : <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>true</isPowerSaverEnabled><sensorId>104</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
```

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 DAS 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 `<DAS_HOME>/repository/deployment/server` to `<DAS_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 the 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 <DAS_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 are sent to this web application and logged.

2. Log into the DAS Management Console. Click **Tools** tab => **Event Simulator** to open the **Event Stream Simulator** page. In the **Event Stream Name** field, select the required event stream. Then click **Play** for the required file in the **Send multiple events** section.

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

The events received by DAS are logged as follows in the DAS console by the generic-log-service web application.

```
Logger service initiated
Event received : <event><metaData><timestamp>199008231226</timestamp><hashcode>23521</hashcode><metaData><correlationData><card manufacturer_id=1001/><card manufacturerer_id=></correlationData><payloadData></payloadData></event></events>
endor>WALMRT</vendor><amount>2221.56</amount><description>NA</description><country code=US</country code><location>PHX</location><card no>4203-9834-2912-1114</card no></payloadData></event></events>
Event received : <event><metaData><timestamp>199008731249</timestamp><hashcode>97988</hashcode><metaData><correlationData><card manufacturerer_id=1001/><card manufacturerer_id=></correlationData><payloadData></payloadData></event></events>
endor>ARMANI</vendor><amount>326.0</amount><description>Armani</description><country code=ITL</country code><location>ROME</location><card no>4203-9834-2912-1114</card no></payloadData></event></events>
Event received : <event><metaData><timestamp>199008131245</timestamp><hashcode>23313</hashcode><metaData><correlationData><card manufacturerer_id=1001/><card manufacturerer_id=></correlationData><payloadData></payloadData></event></events>
endor>BATA</vendor><amount>2321.56</amount><description>NA</description><country code=LK</country code><location>CHB</location><card no>4203-9834-2912-1114</card no></payloadData></event></events>
Event received : <event><metaData><timestamp>199008131242</timestamp><hashcode>33092</hashcode><metaData><correlationData><card manufacturerer_id=1001/><card manufacturerer_id=></correlationData><payloadData></payloadData></event></events>
endor>ZAMZA</vendor><amount>21.56</amount><description>NA</description><country code=LK</country code><location>KANDY</location><card no>4203-9834-2912-1114</card no></payloadData></event></events>
Event received : {"event": {"metaData": {"timestamp": "199008131245"}, "payloadData": {"vendor category": "Leather Products", "country": "LK", "transaction category": "High", "transaction time": "199008031245"}}, "event": {"metaData": {"timestamp": "199008031226"}, "payloadData": {"vendor category": "Department Store", "country": "US", "transaction category": "Low", "transaction time": "19900801031245"}}, "event": {"metaData": {"timestamp": "199008731249"}, "payloadData": {"vendor category": "Fashion", "country": "ITL", "transaction category": "Low", "transaction time": "1290808131245"}}, "event": {"metaData": {"timestamp": "199010131242"}, "payloadData": {"vendor category": "Unknown", "country": "LK", "transaction category": "Low", "transaction time": "109008131245"}}, "event": {"event": "Hello Lasantha Fernando,"}
You have done transaction with your credit card for an amount Rs. 2321.56 with vendor: BATA.

Event received : Hello Sriskandarajah Suhothayan,
You have done transaction with your credit card for an amount Rs. 2221.56 with vendor: WALMRT.

Event received : Hello Mohanadarshan Vivekanandalingam,
You have done transaction with your credit card for an amount Rs. 21.56 with vendor: ZAMZA.

Event received : Hello Rajeev Sampath,
You have done transaction with your credit card for an amount Rs. 326.0 with vendor: ARMANI.
```

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 DAS 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 <DAS_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 <DAS_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 <DAS_HOME>/samples/cep/consumers/axis2-log-service/build.xml file.

This builds the wso2Event client that fetches the published events in the <DAS_HOME>/samples/cep/artifacts/0063/eventsimulatorfiles/events.csv file from the soapendpoint as shown below.

```
-folder.check:
-assign.sample:
-assign.main:
[echo] Sample No :
[echo] Services Dir : ../../repository/deployment/server/axis2services
folder.set:
clean:
init:
[mkdir] Created dir: /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/axis2-log-service/temp
[mkdir] Created dir: /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/axis2-log-service/temp/classes
compile:
[javac] /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/axis2-log-service/build.xml:92: warning: 'includeantruntime' was not set, defaulting to true
build.sysclasspath=last; set to false for repeatable builds
[javac] Compiling 1 source file to /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/axis2-log-service/temp/classes
build-service:
[mkdir] Created dir: /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/axis2-log-service/temp/Axis2LogService
[mkdir] Created dir: /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/axis2-log-service/temp/Axis2LogService/META-INF
[copy] Copying 1 file to /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/axis2-log-service/temp/Axis2LogService/META-INF
[copy] Copying 1 file to /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/axis2-log-service/temp/Axis2LogService
[jar] Building jar: /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/repository/deployment/server/axis2services/Axis2LogService.jar
BUILD SUCCESSFUL
Total time: 1 second
```

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 <DAS_HOME>/samples/cep/artifacts/0063/eventsimulatorfiles/events.csv file to the publisher.

Send multiple events

Input Data by File 		Action
File	Stream Configuration	
events.csv	org.wso2.event.sensor.stream:1.0.0	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>	

5. You view the WSO2 events that are published to the DAS server in the logs of the consumer terminal as shown below.

```
Event received : <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>true</isPowerSaverEnabled><sensorId>101</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
Event received : <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>false</isPowerSaverEnabled><sensorId>103</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
Event received : <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>false</isPowerSaverEnabled><sensorId>100</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
Event received : <events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>true</isPowerSaverEnabled><sensorId>104</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
```

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 `<Das_Home>/repository/conf/output-event-adapters.xml` file as required to send events to the 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">xxxxx</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>
    <!-- Thread Pool Related Properties -->
    <property key="maxThread">100</property>
    <property key="keepAliveTimeInMillis">20000</property>
    <property key="jobQueueSize">10000</property>
</adapterConfig>
```

Building the sample

Start the WSO2 DAS 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 name `emailPublisher` to log the received messages

Executing the sample

This builds the publishes the events in the `<Das_Home>/samples/cep/artifacts/0064/eventsimulatorfi`

les/events.csv file to the emailPublisher endpoint. Follow the steps below to execute the sample.

1. In the DAS Management Console, click to **Main** tab => **Publishers** to open the **Available Event Publishers** page. Then click **Edit** for the existing emailPublisher, and change email property for receiving mail address as shown below.

Event Publisher Configuration

```

3 <!--
4 - Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
5 -
6 - Licensed under the Apache License, Version 2.0 (the "License");
7 - you may not use this file except in compliance with the License.
8 - You may obtain a copy of the License at
9 -
10 -     http://www.apache.org/licenses/LICENSE-2.0
11 -
12 - Unless required by applicable law or agreed to in writing, software
13 - distributed under the License is distributed on an "AS IS" BASIS,
14 - WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15 - See the License for the specific language governing permissions and
16 - limitations under the License.
17 -->
18 <eventPublisher name="emailPublisher" statistics="disable"
19         trace="disable" xmlns="http://wso2.org/carbon/eventpublisher">
20     <from streamName="org.wso2.event.sensor.stream" version="1.0.0"/>
21     <mapping customMapping="disable" type="text"/>
22     <to eventAdapterType="email">
23         <property name="email.address">tharik@wso2.com</property> ←
24         <property name="email.subject">WSO2 CEP - Sensor information</property>
25     </to>
26 </eventPublisher>
27

```

Position: Ln 1, Ch 1 Total: Ln 27, Ch 1175

Toggle editor

Update **Reset**

2. In the **Tools** tab, click **Event Simulator** to open the **Event Simulator** page. In the **Event Stream Name** field, select `org.wso2.event.sensor.stream:1.0.0`. Then click **Play** for the event.csv file in the **Send multiple events** section to send events.

Tools

Home > Tools > Event Simulator

Event Stream Simulator

Send multiple events

File	Stream Configuration	Action
events.csv	org.wso2.event.sensor.stream:1.0.0	Play Configure Delete

Browse... No file selected. **upload**

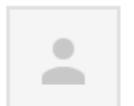
Send single event

Event Stream Name * **select event stream**

Send

Events received by the email receiver are logged as follows.

WSO2 CEP - Sensor information



wso2cloudtest@gmail.com

to me

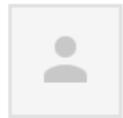
meta_timestamp:199008131285,
meta_isPowerSaverEnabled:true,
meta_sensorId:101,
meta_sensorName:temperature,
correlation_longitude:23.45656,
correlation_latitude:7.12324,
humidity:100.34,
sensorValue:23.4545



wso2cloudtest@gmail.com

to me ▾

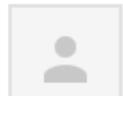
meta_timestamp:199008131345,
meta_isPowerSaverEnabled:false,
meta_sensorId:103,



wso2cloudtest@gmail.com

to me ▾

meta_timestamp:199008131325,
meta_isPowerSaverEnabled:true,
meta_sensorId:102,



wso2cloudtest@gmail.com

to me

meta_timestamp:199008131245,
meta_isPowerSaverEnabled:false,
meta_sensorId:100,

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 <DAS_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>
        <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 DAS by adding relevant jars to support the **SMS transport**.

Building the sample

Start the WSO2 DAS 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 <DAS_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.

```
s m s c s i m . j a r
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 its 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 smscsim.jar:../../lib/smpp.jar com.logica.smssim.Simulator`

4. In the simulator 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.

```
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
- 0 exit
> 1
Enter port number> 2775
Starting listener... started.
```

5. Start the WSO2 DAS 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 <DAS_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv file to the publisher.

Send multiple events

Input Data by File 		Action
File	Stream Configuration	
events.csv	org.wso2.event.sensor.stream:1.0.0	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>	

You view the JSON events that are published by the DAS server in the logs of the simulator as shown below.

```
> 08:08:57 [sys] new connection accepted
08:08:57 [] client request: (bindreq: (pdu: 33 2 0 1) cep1 cep123 52 (addrrang: 0 0 ))
08:08:57 [cep1] authenticated cep1
08:08:57 [cep1] server response: (bindresp: (pdu: 0 0 0000003 0 1) Smsc Simulator)
08:08:57 [cep1] client request: (submit: (pdu: 303 4 0 2) (addr: 0 0 CEP1) (addr: 0 0 +9477117673) (sm: msg: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": false, "sensorId": 100, "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": 100.34, "sensorValue": 23.4545}})) (opt: ))
08:08:57 [cep1] putting message into message store
08:08:57 [cep1] server response: (submit_resp: (pdu: 0 00000004 0 2) Smsc2001 )
08:08:58 [cep1] client request: (submit: (pdu: 302 4 0 3) (addr: 0 0 CEP1) (addr: 0 0 +9477117673) (sm: msg: {"event": {"metaData": {"timestamp": "199008131285", "isPowerSaverEnabled": true, "sensorId": 101, "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": 100.34, "sensorValue": 23.4545}})) (opt: ))
08:08:58 [cep1] putting message into message store
08:08:58 [cep1] server response: (submit_resp: (pdu: 0 00000004 0 3) Smsc2002 )
08:08:59 [cep1] client request: (submit: (pdu: 302 4 0 4) (addr: 0 0 CEP1) (addr: 0 0 +9477117673) (sm: msg: {"event": {"metaData": {"timestamp": "199008131325", "isPowerSaverEnabled": true, "sensorId": 102, "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": 100.34, "sensorValue": 23.4545}})) (opt: ))
08:08:59 [cep1] putting message into message store
08:08:59 [cep1] server response: (submit_resp: (pdu: 0 00000004 0 4) Smsc2003 )
08:09:00 [cep1] client request: (submit: (pdu: 303 4 0 5) (addr: 0 0 CEP1) (addr: 0 0 +9477117673) (sm: msg: {"event": {"metaData": {"timestamp": "199008131345", "isPowerSaverEnabled": false, "sensorId": 103, "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": 100.34, "sensorValue": 23.4545}})) (opt: ))
08:09:00 [cep1] putting message into message store
08:09:00 [cep1] server response: (submit_resp: (pdu: 0 00000004 0 5) Smsc2004 )
```

Enter 5 to view the list of messages published from WSO2 DAS as shown below.

```
| Msg Id | Sender | Servr | Source address | Dest address | Message...
+-----+-----+-----+-----+-----+
| Smsc2004 | [cep1] | [CMT | [CEP1 | [|9477117673 | [{"event": {"metaData": {"timestamp": "199008131345", "isPowerSaverEnabled": false, "sensorId": 103, "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": 100.34, "sensorValue": 23.4545}}}]
| Smsc2003 | [cep1] | [CMT | [CEP1 | [|9477117673 | [{"event": {"metaData": {"timestamp": "199008131325", "isPowerSaverEnabled": true, "sensorId": 102, "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": 100.34, "sensorValue": 23.4545}}}]
| Smsc2002 | [cep1] | [CMT | [CEP1 | [|9477117673 | [{"event": {"metaData": {"timestamp": "199008131285", "isPowerSaverEnabled": true, "sensorId": 101, "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": 100.34, "sensorValue": 23.4545}}}]
| Smsc2001 | [cep1] | [CMT | [CEP1 | [|9477117673 | [{"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": false, "sensorId": 100, "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": 100.34, "sensorValue": 23.4545}}}]
```

Sample 0066 - Publishing JSON Events via MQTT Transport

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

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 DAS 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 DAS 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 `<DAS_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 `<DAS_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 <DAS_HOME>/samples/cep/consumers/mqtt/build.xml file.

This builds the MQTT client which fetches the published events in the <DAS_HOME>/samples/cep/artifacts/0066/eventsimulatorfiles/events.csv file from the eventPublisher endpoint as shown below.

```
Buildfile: /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/mqtt/build.xml

init:
compile:
    [javac] /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/mqtt/build.xml:55: warning: 'includeantruntime' was not set, defaulting to build.sysclasspath.last; set to false for repeatable builds
    [copy] Copying 1 file to /Users/praneesha/Documents/Product_Packs/CEP/CEP-4.0.0/June-18/wso2cep-4.0.0-SNAPSHOT/samples/consumers/mqtt/temp/classes

run:
    [echo] Configure -Durl=<xxx> -Dtopic=<xxx>
    [java] Connected to tcp://localhost:1883
```

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 <DAS_HOME>/samples/cep/artifacts/0066/eventsimulatorfiles/events.csv file to the publisher.

Send multiple events

Input Data by File switch to configure database for simulation		Action
File	Stream Configuration	
events.csv	org.wso2.event.sensor.stream:1.0.0	<input style="margin-right: 5px; border: 1px solid red; border-radius: 5px; padding: 2px 10px;" type="button" value="Play"/> <input style="border: 1px solid #ccc; border-radius: 5px; padding: 2px 10px;" type="button" value="Configure"/> <input style="border: 1px solid #ccc; border-radius: 5px; padding: 2px 10px;" type="button" value="Delete"/>
Choose File <input style="width: 150px;" type="text" value="No file chosen"/> <input style="border: 1px solid #ccc; border-radius: 5px; padding: 2px 10px;" type="button" value="upload"/>		

You view the JSON events that are published to the DAS server in the logs of the consumer terminal as shown below.

```
[java] Received Message -----
[java] | Topic:sensordata
[java] | Message: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": "true", "sensorId": "101", "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": "100.34", "sensorValue": "23.4545"}}}
[java] -----
[java] Received Message -----
[java] | Topic:sensordata
[java] | Message: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": "false", "sensorId": "103", "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": "100.34", "sensorValue": "23.4545"}}}
[java] -----
[java] Received Message -----
[java] | Topic:sensordata
[java] | Message: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": "true", "sensorId": "104", "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": "100.34", "sensorValue": "23.4545"}}}
[java] -----
[java] Received Message -----
[java] | Topic:sensordata
[java] | Message: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": "false", "sensorId": "100", "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": "100.34", "sensorValue": "23.4545"}}}
```

Sample 0067 - Publishing Map Events via Cassandra Transport

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

Introduction

This sample demonstrates how to publish incoming events to Apache Cassandra using the Cassandra event publisher of DAS. 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 DAS 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 <DAS_HOME>/repository/deployment/server to <DAS_HOME>/samples/cep/artifacts/0067.
- Creates an event stream named org.wso2.event.transaction.stream:1.0.0.
- Loads the events.csv file and related configuration files so that the 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 publishes 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.

- Cluster Name: Test Cluster
- Keyspace: CEP_KS
- Column Family: CF_Transactions (Comparator should be of type 'UTF8Type' or 'AsciiType')
- Connection Username: admin
- 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 DAS_KS;
- Use keyspace: use DAS_KS admin 'admin';
- Create column family: create column family CF_Transactions with comparator = 'UTF8Type' and caching='ALL' ;

Create a New Event Publisher

Enter Event Publisher Details	
Event Publisher Name*	cassandra <small>Enter a unique name to identify Event Publisher</small>
From	
Event Source*	org.wso2.event.transaction.stream:1.0.0 <small>The stream of events that need to be published</small>
Stream Attributes	meta_timestamp long, meta_hashcode int, correlation_card_manufacturer_id long, vendor string, amount double, description string, country_code string, location string, card_no string
To	cassandra <small>Select the type of Adapter to publish events</small>
Static Adapter Properties	
Hosts *	localhost <small>Hostnames or ipaddresses separated by comma e.g., testhost1,testhost2</small>
Port	9160 <small>The cassandra port, if not defined default port will be used</small>
User Name	admin
Password	*****
Keypspace Name *	CEP_KS
Column Family Name *	CF_Transactions
Strategy Class	<small>The strategy of the keyspace, if not defined 'org.apache.cassandra.locator.SimpleStrategy' will be used</small>
Replication Factor	<small>The replication factor of keyspace, if not defined '1' will be used</small>
Indexed Columns	<small>Columns to be indexed, separated by comma e.g., key1,key2. Index of type "KEYS" with name "{keyspaceName}_{columnFamilyName}_{columnKey}_Index" will be applied to the columns</small>
Mapping Configuration	
Message Format*	map <small>Select the output message format</small>
Advanced	
Add Event Publisher	

2. Log into the DAS Management Console. In the **Tools** tab, click **Event Simulator** to open the **Event Simulator** page. In the **Event Stream Name** field, select org.wso2.event.transaction.stream:1.0.0. In the **Sen**

d multiple events section, click **Play** for the event .csv file to play the events.

Home > Tools > Event Simulator

Help

Event Stream Simulator

Send single event

Event Stream Name *	<input type="text" value="org.wso2.event.transaction.stream:1.0.0"/>
Stream Attributes	
Meta Attributes	
timestamp(<i>long</i>) *	<input type="text"/>
hashcode(<i>int</i>) *	<input type="text"/>
Correlation Attributes	
card_manufacturer_id(<i>long</i>) *	<input type="text"/>
Payload Attributes	
vendor(<i>string</i>) *	<input type="text"/>
amount(<i>double</i>) *	<input type="text"/>
description(<i>string</i>) *	<input type="text"/>
country_code(<i>string</i>) *	<input type="text"/>
location(<i>string</i>) *	<input type="text"/>
card_no(<i>string</i>) *	<input type="text"/>
<input type="button" value="Send"/> <input type="button" value="Clear"/>	

Send multiple events

Input Data by File switch to configure database for simulation			
File	Stream Configuration	Delay between events(ms)	Action
<input type="text" value="events.csv"/>	org.wso2.event.transaction.stream:1.0.0	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>		

3. You can see the events getting received by DAS by executing the command 'list CF_Transactions;' in Cassandra CLI.

```
[admin@CEP KS] list CF_Transactions;
Using default limit of 100
Using default cell limit of 100
-----
RowKey: 65323738373430342d366561372d346363382d626265612d396463383134623432363531
=> (name=amount, value=323332312e3536, timestamp=1431949637894000)
=> (name=card_no, value=343230332d393833342d323931322d31313134, timestamp=1431949638050000)
=> (name=correlation_card_manufacturer_id, value=31303031, timestamp=1431949638053000)
=> (name=country_code, value=4c4b, timestamp=1431949638054000) Stream Configuration
=> (name=description, value=4e41, timestamp=1431949638055000)
=> (name=location, value=434d42, timestamp=1431949638056000)
=> (name=meta_hashcode, value=323839331, timestamp=1431949638058000) event.transaction.stream:1.0.0
=> (name=meta_timestamp, value=31393930303831331323435, timestamp=1431949638059000)
=> (name=vendor, value=42415441, timestamp=1431949638060000) upload
-----
RowKey: 35313933336634382d653738612d346263622d613334322d333937376530326663336662
=> (name=amount, value=3332362e30, timestamp=1431949638070001)
=> (name=card_no, value=343230332d393833342d323931322d31313134, timestamp=1431949638071000)
=> (name=correlation_card_manufacturer_id, value=31303031, timestamp=1431949638072000)
=> (name=country_code, value=4944c, timestamp=1431949638072001) selected event.stream
=> (name=description, value=41726d610e69, timestamp=1431949638073000)
=> (name=location, value=524f4d45, timestamp=1431949638074000)
=> (name=meta_hashcode, value=3937383938, timestamp=1431949638074001)
=> (name=meta_timestamp, value=31393930303837331323439, timestamp=1431949638075000)
=> (name=vendor, value=41524d414e49, timestamp=1431949638076000)
-----
RowKey: 66616536623230392d393561312d346162332d616433392d643837663030373562376633
=> (name=amount, value=32312e3536, timestamp=1431949638077000)
=> (name=card_no, value=343230332d393833342d323931322d31313134, timestamp=1431949638077001)
=> (name=correlation_card_manufacturer_id, value=31303031, timestamp=1431949638078000)
=> (name=country_code, value=4c4b, timestamp=1431949638079000)
=> (name=description, value=4e41, timestamp=1431949638079001)
=> (name=location, value=4b414e4459, timestamp=1431949638080000)
=> (name=meta_hashcode, value=3833383933, timestamp=1431949638081000)
=> (name=meta_timestamp, value=31393930313031331323432, timestamp=1431949638082000)
=> (name=vendor, value=5a414d5a41, timestamp=1431949638082001)
-----
RowKey: 36616536306136342d663832622d343563322d396135302d653131643034316337313661
=> (name=amount, value=323232312e3536, timestamp=1431949638061000)
=> (name=card_no, value=343230332d393833342d323931322d31313134, timestamp=1431949638062000)
=> (name=correlation_card_manufacturer_id, value=31303031, timestamp=1431949638063000)
=> (name=country_code, value=5553, timestamp=1431949638064000)
=> (name=description, value=4e41, timestamp=1431949638066000)
=> (name=location, value=504858, timestamp=1431949638067000)
=> (name=meta_hashcode, value=3233353231, timestamp=1431949638067001)
=> (name=meta_timestamp, value=31393930303832331323236, timestamp=1431949638068000)
=> (name=vendor, value=57414c4d5254, timestamp=1431949638070000)

4 Rows Returned.
Elapsed time: 76 msec(s).
```

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 DAS 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](#).

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

```
[2015-06-24 16:40:16.084] INFO Verifying properties (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:187] INFO Property broker.id is overridden to 0 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:187] INFO Property log.cleaner.enable is overridden to false (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:188] INFO Property log.dirs is overridden to /tmp/kafka-logs (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:188] INFO Property log.retention.check.interval.ms is overridden to 300000 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:188] INFO Property log.retention.hours is overridden to 168 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:188] INFO Property log.segment.bytes is overridden to 1073741824 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:188] INFO Property num.io.threads is overridden to 8 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:188] INFO Property num.network.threads is overridden to 3 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:189] INFO Property num.recovery.threads.per.data.dir is overridden to 1 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:189] INFO Property port is overridden to 9092 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:189] INFO Property socket.receive.buffer.bytes is overridden to 102400 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:189] INFO Property socket.request.max.bytes is overridden to 104857600 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:189] INFO Property socket.send.buffer.bytes is overridden to 102400 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:189] INFO Property zookeeper.connect is overridden to localhost:2181 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:189] INFO Property zookeeper.connection.timeout.ms is overridden to 6000 (kafka.utils.VerifiableProperties)
[2015-06-24 16:40:189] INFO [kafka Server 0], starting (kafka.server.KafkaServer)
[2015-06-24 16:40:189] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2015-06-24 16:40:189] INFO Client environment:zookeeper.version=3.4.6-1569965, built on 02/29/2014 09:09 GMT (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:host.name=phoenix (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.version=1.7_0.80 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.vendor.version=1.7.0-oracle-jre (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.home=/usr/lib/jvm/java-7-oracle/jre (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.class.path=/home/thilina/Software/kafka_2.10-0.8.2.1/bin/../core/lib/dependant-libbs-2.10.4*.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/../core/lib/kafka-examples.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..../contrib/hadoop-consumer/build/libs/kafka-hadoop-consumer.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..../clients/build/libs/kafka-clients*.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/kafka_2.10-0.8.2.1-test.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/kafka_2.10-0.8.2.1.1-test.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/kafka_2.10-0.8.2.1.1-scaladoc.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/kafka_2.10-0.8.2.1.1-javadoc.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/kafka_2.10-0.8.2.1.1-test.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/kafka_2.10-0.8.2.1.1-sources.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/kafka_2.10-0.8.2.1.1-bin/..../libs/metrics-core-2.0.2.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/metrics-core-2.0.2.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/scal-library-2.10.4.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/slf4j-log4j12-1.6.1.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/slf4j-log4j12-1.6.1.1.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/snappy-java-1.1.1.6.jar:/home/thilina/Software/kafka_2.10-0.8.2.1-bin/..../libs/zkclient-0.3.jar:/home/thilina/Software/zookeeper-3.4.6.jar:/home/thilina/Software/kafka_2.10-0.8.2.1/bin/..../core/build/libs/kafka_2.10-0.8.2.1.jar (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.library.path=/usr/java/packages/lib/amd64:/lib64:/lib64:/lib:/usr/lib (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.io.tmpdir=/tmp (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.compiler=NONE (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:java.awt.headless=true (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:os.arch=amd64 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:os.version=3.13.6-55-generic (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:user.name=thilina (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:user.home=/home/thilina (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Client environment:user.dir=/home/thilina/Software/kafka_2.10-0.8.2.1 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:189] INFO Initiating client connection, connectString=localhost:2181 sessionTimeout=60000 watcher=org.IoTec.ZkClient@6119e2c1 (org.apache.zookeeper.ZooKeeper)
[2015-06-24 16:40:158] INFO Opening socket connection to server localhost/[27.0.0.1]:2181. Will not attempt to authenticate using SASL (unknown error) (org.apache.zookeeper.ClientCnxn)
[2015-06-24 16:40:161] INFO Socket connection established to localhost/[27.0.0.1]:2181, initiating session (org.apache.zookeeper.ClientCnxn)
[2015-06-24 16:40:166] INFO Session establishment complete on server localhost/[27.0.0.1]:2181, sessionid = 0x1e2452aa40000, negotiated timeout = 6000 (org.apache.zookeeper.ClientCnxn)
[2015-06-24 16:40:167] INFO zookeeper state changed (SyncConnected) (org.OItec.ZkClient.ZkClient)
[2015-06-24 16:40:180] INFO Loading logs. (kafka.log.LogManager)
[2015-06-24 16:40:16,321] INFO Completed load of log sensorInfo-0 with log end offset 0 (kafka.log.Log)
[2015-06-24 16:40:16,329] INFO Completed load of log sensorStream-0 with log end offset 39 (kafka.log.Log)
[2015-06-24 16:40:16,332] INFO Completed load of log sensorData-0 with log end offset 12 (kafka.log.Log)
[2015-06-24 16:40:16,355] INFO Completed load of log filteredsensorStream-0 with log end offset 0 (kafka.log.Log)
[2015-06-24 16:40:16,357] INFO Loading log complete. (kafka.log.LogManager)
[2015-06-24 16:40:16,357] INFO Starting log cleanup with a period of 1000 ms. (kafka.log.LogManager)
[2015-06-24 16:40:16,359] INFO Starting log flushes with a defualt period of 223372836854775907 ms. (kafka.log.LogManager)
[2015-06-24 16:40:16,356] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.Acceptor)
[2015-06-24 16:40:16,357] INFO [Socket Server on Broker 0], Started (kafka.network.SocketServer)
[2015-06-24 16:40:16,397] INFO Will not load MX4J, mx4j-tools.jar is not in the classpath (kafka.utils.Mx4jLoader$)
[2015-06-24 16:40:16,426] INFO 0 successfully elected as leader (kafka.server.ZookeeperLeaderElector)
[2015-06-24 16:40:16,648] INFO New leader is 0 (kafka.server.ZookeeperLeaderElector$LeaderChangelistener)
[2015-06-24 16:40:16,650] INFO Registered broker 0 at path /brokers/ids/0 with address phoenix:9092. (kafka.utils.ZkUtils$)
[2015-06-24 16:40:16,657] INFO [kafka Server 0], started (kafka.server.KafkaServer)
[2015-06-24 16:40:16,755] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions [filteredSensorStream,0],[sensorInfo,0],[sensorData,0],[sensorStream,0] (kafka.server.ReplicaFetcherManager)
[2015-06-24 16:40:16,785] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions [filteredSensorStream,0],[sensorInfo,0],[sensorData,0],[sensorStream,0] (kafka.server.ReplicaFetcherManager)
```

Building the sample

Start the WSO2 DAS 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 `<Das_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 DAS 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 <DAS_HOME>/samples/cep/artifacts/0058/eventsimulatorfiles/events.csv file to the publisher.

[Send multiple events](#)

Input Data by File switch to configure database for simulation		Action
File	Stream Configuration	
events.csv	org.wso2.event.sensor.stream:1.0.0	Play Configure Delete
Choose File No file chosen	upload	

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 DAS server in the logs of the Kafka consumer terminal as shown below.

```
<events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>false</isPowerSaverEnabled><sensorId>100</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>-7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
<events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>true</isPowerSaverEnabled><sensorId>104</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>-7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
<events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>false</isPowerSaverEnabled><sensorId>103</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>-7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
<events><event><metaData><timestamp>199008131245</timestamp><isPowerSaverEnabled>true</isPowerSaverEnabled><sensorId>101</sensorId><sensorName>temperature</sensorName></metaData><correlationData><longitude>23.45656</longitude><latitude>-7.12324</latitude></correlationData><payloadData><humidity>100.34</humidity><sensorValue>23.4545</sensorValue></payloadData></event></events>
```

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 Web Socket 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

Before starting the output websocket event publisher configuration, set up the [prerequisites required for all CEP samples](#).

Building the sample

Start the WSO2 DAS 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 <DAS_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 <DAS_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 <DAS_HOME>/samples/cep/consumers/websocket/build.xml file.

This builds the WebSocket client which fetches the published events in the <DAS_HOME>/samples/cep/artifacts/0069/eventsimulatorfiles/events.csv file from the wsPublisher endpoint as shown below.

```
run:
[echo] For server mode, configure -Dhost='server host' (optional), -Dport='server port' (optional)
[echo] For client mode, configure -Durl='server url'
[java] Starting Websocket receiver on Server Mode
[java] 2015-09-14 20:29:21.491:INFO::main: Logging initialized @117ms
[java] 2015-09-14 20:29:21.594:INFO:oejs.Server:Thread-1: jetty-9.2.7.v20150116
[java] 2015-09-14 20:29:21.820:INFO:oejs.ContextHandler:Thread-1: Started o.e.j.s.ServletContextHandler@61b3a3ca{/null,AVAILABLE}
[java] 2015-09-14 20:29:21.826:INFO:oejs.ServerConnector:Thread-1: Started ServerConnector@7b5a8198{HTTP/1.1}{localhost:9099}
[java] 2015-09-14 20:29:21.826:INFO:oejs.Server:Thread-1: Started @455ms
```

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 <DAS_HOME>/samples/cep/artifacts/0069/eventsimulatorfiles/events.csv file to the publisher.

[Send multiple events](#)

Input Data by File switch to configure database for simulation		Action
File	Stream Configuration	
events.csv	org.wso2.event.sensor.stream:1.0.0	Play Configure Delete
Choose File No file chosen	upload	

You view the WSO2 events that are published to the DAS server in the logs of the consumer terminal as shown below.

```
[java] [qtp1274370218-17] INFO org.wso2.carbon.sample.websocket.WebSocketServer - Server Received TEXT message: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": false, "sensorId": "01", "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": "100.34", "sensorValue": "23.4545"}}}
```

```
[java] [qtp1274370218-14] INFO org.wso2.carbon.sample.websocket.WebSocketServer - Server Received TEXT message: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": true, "sensorId": "01", "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": "100.34", "sensorValue": "23.4545"}}}
```

```
[java] [qtp1274370218-10] INFO org.wso2.carbon.sample.websocket.WebSocketServer - Server Received TEXT message: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": false, "sensorId": "04", "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": "100.34", "sensorValue": "23.4545"}}}
```

```
[java] [qtp1274370218-17] INFO org.wso2.carbon.sample.websocket.WebSocketServer - Server Received TEXT message: {"event": {"metaData": {"timestamp": "199008131245", "isPowerSaverEnabled": true, "sensorId": "04", "sensorName": "temperature"}, "correlationData": {"longitude": "23.45656", "latitude": "7.12324"}, "payloadData": {"humidity": "100.34", "sensorValue": "23.4545"}}}
```

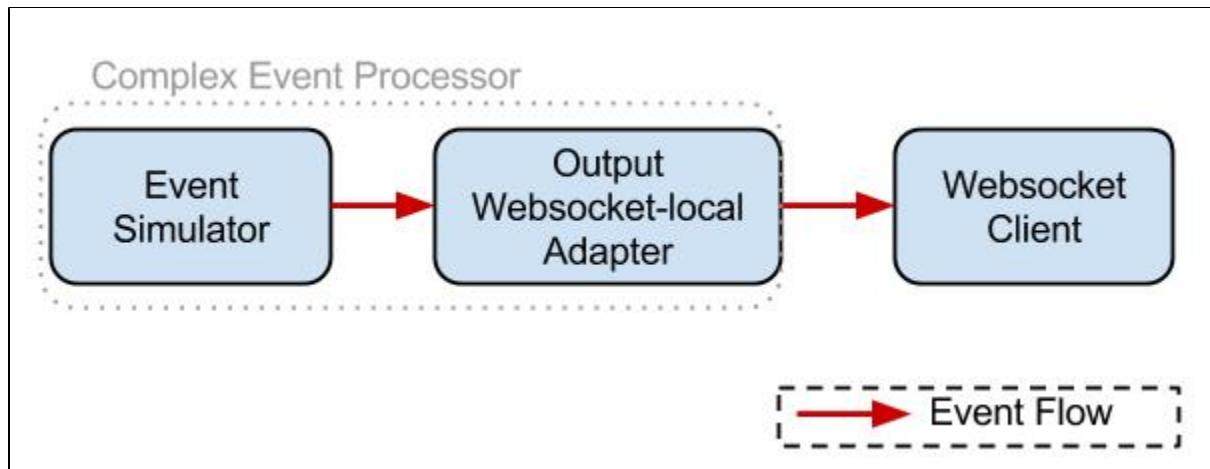
Sample 0070 - Publishing JSON Events via Websocket-Local Output Event Adapter

- [Introduction](#)
- [Prerequisites](#)
- [Starting sample CEP configurations](#)
- [Connect to DAS via a Websocket client \(to receive events\)](#)

Introduction

This sample demonstrates how the WSO2 DAS can publish events to a Websocket client.

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



The following is a summary of the steps that are followed to run this sample.

- Step 1: Start the DAS server. This starts an inbuilt Websocket server in DAS.
- Step 2: Define an Event Stream in DAS.
- Step 3: Create an Output Websocket-local Event Adapter so that events of the type defined in step 1 are published via the DAS Websocket server.
- Step 4: Connect a Websocket client to the Websocket Server in DAS.
- Step 5: Send a few events to the DAS. The inbuilt Event Simulator tool in DAS is used for this purpose. The events sent are received by the Websocket client.

Prerequisites

See [Prerequisites](#) in the CEP Samples Setup page.

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

```
ant -Dsn=0070
```

This copies the `outputwebsocket.war` webapp to `<DAS_HOME>/sample/artifacts/0070/webapps` directory.

Starting sample CEP configurations

Start the WSO2 DAS server with the sample configuration numbered 0070. For instructions, see [Starting sample CEP configurations](#).

This sample configuration points the default Axis2 repo to `<DAS_HOME>/samples/cep/artifacts/0070` (by default, the Axis2 repo is `<DAS_HOME>/repository/deployment/server`). As a result, the artifacts in `<DAS_HOME>/sample/artifacts/0070` directory are deployed.

When these sample configurations are deployed:

- An event stream is defined.
- The event simulator is available to play a set of events to DAS.
- An Output Websocket-local Event Adapter will be created.

In other words, all the first three steps listed in Introduction section are completed 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.

Connect to DAS 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> 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:

```

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 DAS 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.



The screenshot shows a browser's developer tools interface with the 'Console' tab selected. The tab bar includes tabs for Net, CSS, JS, Security, Logging, and Clear. The console output area displays the following JavaScript code:

```

» 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.

The screenshot shows the 'Browser Console' window with the 'JS' tab selected. The console output is as follows:

```

09:34:51.313 < var ws = new WebSocket("ws://localhost:9763/outputwebsocket/wsLocalOutputAdapter");
09:34:51.313     ws.onopen = function() {
09:34:51.313         console.log("opened");
09:34:51.313     };
09:34:51.313
09:34:51.313     ws.onmessage = function (evt) {
09:34:51.313         alert("Message: " + evt.data);
09:34:51.313     };
09:34:51.313
09:34:51.313     ws.onclose = function() {
09:34:51.313         console.log("closed!");
09:34:51.313     };
09:34:51.313
09:34:51.313     ws.onerror = function(err) {
09:34:51.313         console.log("Error: " + err);
09:34:51.313     };
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 DAS.

Step III: Next and final step is to send four events to the DAS. We will send the following four events of type `org.wso2.event.sensor.stream` to the DAS.

```

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

```

The Event Simulator tool in DAS is used to send events.

To use the Event Simulator, log in to the DAS Management Console and go to **Tools --> Event Simulator**.

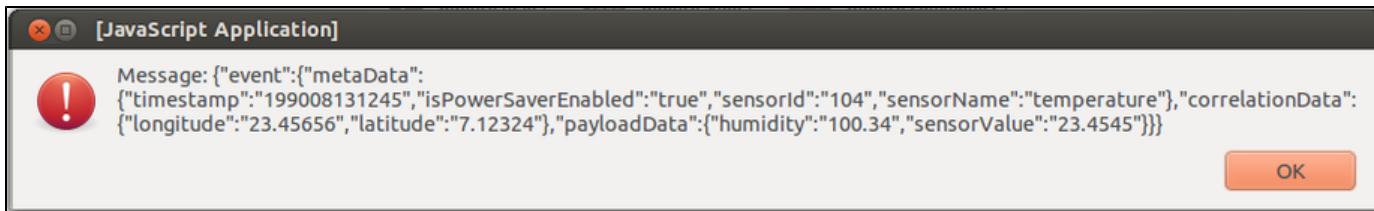
In the **Event Simulator** page, an input data file ready to be played is displayed in the **Send Multiple Events** section. Click **Play** as shown in the screenshot below to send the events included in this data file to the DAS.

The screenshot shows the 'Send multiple events' interface. It has a table with the following data:

File	Stream Configuration	Delay between events(ms)	Action
events.csv	org.wso2.event.sensor.stream:1.0.0	click the configure button	Play (button highlighted with a red box)

Below the table are two buttons: 'Browse...' and 'upload'.

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.



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 use any execution plans.

Prerequisites

Set up the prerequisites required for all samples.

Building the sample

Start the WSO2 DAS 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 `<Das_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.

Configuration	
Stream Name :	<input type="text" value="org.wso2.event.sensor.stream"/>
CEP Host Name :	<input type="text" value="localhost"/>
Mode :	<input type="text" value="WEBSOCKET"/>
Version :	<input type="text" value="1.0.0"/>
CEP Port :	<input type="text" value="9443"/>
Polling Interval :	<input type="text" value="30"/>
<input type="button" value="Connect"/>	<input type="button" value="Clear All"/>

The fields in the form are described in the table below.

Field Name	Description	Default Value
Stream Name	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.	org.wso2.event.sensor.stream
Version	The version of the event stream to which you want to subscribe.	1.0.0
CEP Host	The host on which the CEP server (which contains the UI publisher) runs.	localhost
CEP Port	The port on which the CEP server (which contains the UI publisher) runs.	9443
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.	WEBSOCKET
Polling Interval	The time interval at which events should be polled from the UI publisher. This time interval is specified in seconds. This field value is only applicable if the value specified in the Mode field is HTTP .	30

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 <DAS_HOME>/samples/cep/artifacts/0071/eventsimulatorfiles/events.csv file to the publisher.

[Send multiple events](#)

Input Data by File switch to configure database for simulation		Action
File	Stream Configuration	
events.csv	org.wso2.event.sensor.stream:1.0.0	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>	

The output events published from WSO2 CEP and received by the client Jaggery application are logged as follows.

```

Received Events

org.wso2.event.sensor.stream:1.0.0 =
[1473169499030,"199008131245","false","100","temperature","23.45656","7.12324","100.34","23.4545"]
org.wso2.event.sensor.stream:1.0.0 =
[1473169500035,"199008131245","true","101","temperature","23.45656","7.12324","100.34","23.4545"]
org.wso2.event.sensor.stream:1.0.0 =
[1473169501036,"199008131245","false","103","temperature","23.45656","7.12324","100.34","23.4545"]
org.wso2.event.sensor.stream:1.0.0 =
[1473169502036,"199008131245","true","104","temperature","23.45656","7.12324","100.34","23.4545"]

Clear

```

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 <DAS_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 <DAS_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.

```
<H2DatabaseConfiguration>
<property name="web" />
<property name="webPort">8082</property>
<property name="webAllowOthers" />
</H2DatabaseConfiguration>
```

Starting the server

Start the WSO2 DAS server with the sample configuration numbered 0072. For instructions on starting WSO2 DAS server with a sample configuration, see [Starting sample CEP configurations](#). This sample configuration points the default Axis2 repository to <DAS_HOME>/samples/cep/artifacts/0072/ directory. (By default, the Axis2 repository is pointed to <DAS_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 <DAS_HOME>/samples/cep/artifacts/0072/eventsimulatorfiles/events.csv file to the publisher.

Send multiple events

Input Data by File	switch to configure database for simulation	Stream Configuration	Delay between events(ms)	Action
events.csv		org.wso2.event.sensor.stream:1.0.0	1000	Play Configure Delete
Choose File No file chosen	upload			

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.

Login

Saved Settings:	Generic H2 (Embedded)
Setting Name:	<input type="text" value="Generic H2 (Embedded)"/> Save Remove
Driver Class:	<input type="text" value="org.h2.Driver"/>
JDBC URL:	<input type="text" value="jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_DELAY=-1"/>
User Name:	<input type="text" value="wso2carbon"/>
Password:	<input type="password" value="*****"/>
Connect Test Connection	

Configuration	Description
Driver Class	Enter the value as org.h2.Driver
JDBC URL	Enter the URL of the database. For example: jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_DELAY=-1

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 screenshot shows the WSO2 Data Analytics Server interface. On the left, there is a tree view of the database schema under 'jdbc:h2:repository/database/WSO2CARBON'. The 'SENSORDATA' table is listed under the 'UM' category. On the right, there is a 'Run (Ctrl+Enter)' button, a 'Clear' button, and a 'SQL statement' input field. Below these are several 'Important Commands' buttons: 'Displays this Help Page', 'Shows the Command History', 'Executes the current SQL statement', and 'Disconnects from the database'. Under 'Sample SQL Script', there is a code snippet for creating a table named 'TEST' with columns 'ID' (INT PRIMARY KEY) and 'NAME' (VARCHAR(255)). It includes INSERT statements for 'Hello' and 'World', a SELECT query, an UPDATE query, and a DELETE query. At the bottom, there is a 'Help ...' link and an 'Adding Database Drivers' section with instructions for registering database drivers.

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.

SELECT * FROM SENSORDATA;								
CORRELATION_LATITUDE	CORRELATION_LONGITUDE	HUMIDITY	META_ISPOWERSAVERENABLED	META_SENSORID	META_SENSORNAME	META_TIMESTAMP	SENSORVALUE	
7.12324	23.45656	100.33999633799062	FALSE	100	temperature	1.99008125E11	23.4545	
7.12324	23.45656	100.33999633799062	TRUE	101	temperature	1.99008125E11	23.4545	
7.12324	23.45656	100.33999633799062	FALSE	103	temperature	1.99008125E11	23.4545	
7.12324	23.45656	100.33999633799062	TRUE	104	temperature	1.99008125E11	23.4545	

(4 rows, 12 ms)

Sending Notifications Through Published Events Using Spark

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

Introduction

This sample demonstrates how you can send notifications through events published from WSO2 DAS using Apache Spark. The notifications are sent to alert about records of an existing table in the Spark environment satisfying a defined condition(s). This sample involves creating a table with a few product names and quantities in the DAL, and

sending notifications when the quantity of a product falls below a defined value.

Prerequisites

Set up the general prerequisites required for WSO2 DAS.

Building the sample

Follow the steps below to build the sample.

Creating the receiving event stream and the table in DAL

Follow the steps below to create a table named PRODUCTS together with the receiving event stream in the Data Access Layer (DAL).

1. Log in to the DAS management console using the following URL: https://<DAS_HOST>:<DAS_PORT>/carbon/
2. Click **Main**, and then click **Streams**.
3. Click **Add Event Stream**.
4. Enter the values as shown below to create an event stream named PRODUCTS_STREAM with two attributes as product name and quantity. For more information on creating event streams, see [Understanding Event Streams and Event Tables](#).

Home > Manage > Event > Streams

Define New Event Stream

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	PRODUCTS_STREAM <small>Name of the Event Stream</small>
Event Stream Version*	1.0.0 <small>Version of the event stream (Eg : 1.0.0)</small>
Event Stream Description	<small>Description of the event stream</small>
Event Stream Nick-Name	<small>Nick of the event stream</small>

Stream Attributes

Meta Data Attributes
No meta data attributes are defined

Attribute Name : Attribute Type :

Correlation Data Attributes
No correlation data attributes are defined

Attribute Name : Attribute Type :

Payload Data Attributes

Attribute Name	Attribute Type	Actions
product-name	string	Delete
quantity	int	Delete

Attribute Name : Attribute Type :

[Add Event Stream](#) [Next \[Persist Event\]](#)

5. Click **Next (Persist Event)**.
6. Enter the values as shown below in the next screen to persist the created event stream. For more information on creating event streams, see [Persisting Event Streams](#).

Enter Event Index Details

Persist Event Stream

Record Store
EVENT_STORE

Meta Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
No meta data attributes are defined					

Correlation Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
No correlation data attributes are defined					

Payload Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
<input checked="" type="checkbox"/>	product-name	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	quantity	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes

No arbitrary data attributes are defined

Attribute Name : Attribute Type : Primary Key : Index Column : Score Param :

7. Click **Save Event Stream**.

Sending events to the receiving event stream

Follow the steps below to simulate the sending of events to the created receiving event stream.

1. Log in to the DAS management console using the following URL, if you are not already logged in: https://<DAS_HOST>:<DAS_PORT>/carbon/
2. Click **Tools**, and then click **Event Simulator**.
3. Upload the events to be sent to it in a CSV file (e.g. **footwear.csv**), and click **Configure** as shown below.

Send multiple events

Input Data by File

File	Stream Configuration	Delay between events(ms)	Action
footwear.csv	click the configure button	click the configure button	<input type="button" value="Configure"/> <input type="button" value="Delete"/>

No file chosen

4. Enter the details as shown below, and click **Configure**.

Event Mapping Configuration

File name	footwear.csv
Select the target event stream*	<input type="button" value="PRODUCTS_STREAM:1.0.0"/>
Field delimiter*	,
Delay between events in milliseconds*	0

5. Click **Play** in the next screen as shown below.

Send multiple events

Input Data by File

File	Stream Configuration	Delay between events(ms)	Action
footwear.csv	PRODUCTS_STREAM:1.0.0	1000	<input type="button" value="Play"/> <input type="button" value="Configure"/> <input type="button" value="Delete"/>

No file chosen

6. Click **Main**, and then click **Data Explorer**. Select the name of the table to view it as shown below.

The screenshot shows the 'Data Explorer' section of the WSO2 Data Analytics Server management console. At the top, there's a search bar with fields for 'Table Name*' (set to 'PRODUCTS_STREAM'), 'Maximum Result Count' (set to '1000'), and a 'Schedule Data Purging' button. Below the search bar are options to search by 'Date Range', 'Primary Key', or 'Query'. A 'Search' button is highlighted in blue.

The main area is titled 'Results' and displays a note: 'Note: Total record count for the table is not available.' Below this is a table titled 'PRODUCTS_STREAM' with three columns: 'product-name', 'quantity', and 'timestamp'. The data shows four rows:

	product-name	quantity	timestamp
1	Nike	7	2016-01-07 12:01:58 IST
2	Adidas	3	2016-01-07 12:01:59 IST
3	Puma	2	2016-01-07 12:02:00 IST
4	Bettans	0	2016-01-07 12:02:01 IST

At the bottom of the table, there are navigation links ('<< < ... > >>'), a page number ('Go to page: 1'), a row count ('Row count: 10'), and a note 'Showing 1-10 of 1000'.

Creating the corresponding table in Apache Spark

Follow the steps below to create a virtual table in the Apache Spark environment within WSO2 DAS to map the PRODUCTS_STREAM table in the DAL.

1. Log in to the DAS management console using the following URL, if you are not already logged in: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Main**, and then click **Console**.
3. Enter the following Spark SQL query in the Spark console, and press Enter key to execute it.

```
CREATE TEMPORARY TABLE PRODUCTS_MASTER
USING CarbonAnalytics
OPTIONS (tableName "PRODUCTS_STREAM",
        schema "product-name STRING,quantity INT");
```

Creating the event stream to publish event from Spark

For publishing events from Spark, you have to define an event stream with given stream attributes that will be published from Spark.

1. Log in to the DAS management console using the following URL: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Main**, and then click **Streams**.
3. Click **Add Event Stream**.
4. Enter the values as shown below to create an event stream named PRODUCT_ALERTS_STREAM with two attributes as product name and quantity. For more information on creating event streams, see [Event Streams](#).

Home > Manage > Event > Streams

Define New Event Stream

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	PRODUCT_ALERTS_STREAM ⑦ <i>Name of the Event Stream</i>		
Event Stream Version*	1.0.0 ⑦ <i>Version of the event stream (Eg : 1.0.0)</i>		
Event Stream Description	⑦ <i>Description of the event stream</i>		
Event Stream Nick-Name	⑦ <i>Nick name of the event stream</i>		

Stream Attributes

Meta Data Attributes
No meta data attributes are defined

Attribute Name :	Attribute Type :	int	Add
------------------	------------------	-----	-----

Correlation Data Attributes
No correlation data attributes are defined

Attribute Name :	Attribute Type :	int	Add
------------------	------------------	-----	-----

Payload Data Attributes

Attribute Name	Attribute Type	Actions
product-name	string	Delete
quantity	int	Delete

Attribute Name :	Attribute Type :	int	Add
------------------	------------------	-----	-----

[Add Event Stream](#) [Next \[Persist Event\]](#)

- Click **Add Event Stream**.

Creating the event receiver for event stream

Once you define the event stream, you need to create an event receiver of the WSO2Event type to receive events from Spark. For more information, see [Configuring Event Receivers](#).

- Log in to the DAS management console using the following URL: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
- Click **Main**, and then click **Receivers**.
- Click **Add Event Receiver**.

4. Enter the values as shown below to create an event receiver of the WSO2Event type for above PRODUCT_ALERTS_STREAM. For more information on creating event streams, see [Event Streams](#).

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="SampleEventReceiver"/> <div style="font-size: small; margin-top: -10px;">⑦ Enter a unique name to identify Event Receiver</div>	
From	
Input Event Adapter Type* <input type="text" value="wso2event"/> <div style="font-size: small; margin-top: -10px;">⑦ Select the type of Adapter to receive events</div>	Following url formats are used to receive events For load-balancing: use "," to separate values for multiple endpoints Eg: {tcp://<hostname>:<port>},tcp://<hostname>:<port>, ...} For failover: use " " to separate multiple endpoints Eg: {tcp://<hostname>:<port>} {tcp://<hostname>:<port>} ...} For more than one cluster: use "{}" to separate multiple clusters Eg: {tcp://<hostname>:<port>} {tcp://<hostname>:<port>} ...},{tcp://<hostname>:<port>} {tcp://<hostname>:<port>} ...}
Usage Tips Ports available for Thrift protocol - TCP port:7611 or SSL port:7711 Ports available for Binary protocol - TCP port:9611 or SSL port:9711	
Adapter Properties	
Is events duplicated in cluster <input checked="" type="checkbox" value="false"/> <div style="font-size: small; margin-top: -10px;">⑦ This depends on how events are published to the server, 'true' only if multiple publishers publish to the same cluster.</div>	
To	
Event Stream* <input type="text" value="PRODUCT_ALERTS_STREAM:1.0.0"/> <div style="font-size: small; margin-top: -10px;">⑦ The event stream that will be generated by the received events</div>	
Mapping Configuration	
Message Format* <input type="text" value="wso2event"/> <div style="font-size: small; margin-top: -10px;">⑦ Select the input message format</div>	
+ Advanced	
<input type="button" value="Add Event Receiver"/>	

5. Click **Add Event Receiver**

Configuring a publisher

You can attach an event publisher such as email or JMS to the PRODUCT_ALERTS_STREAM, and get the events delivered to a preferred location. Follow the steps below to configure a publisher to publish output events from WSO2 DAS, and to send notifications as logs of the terminal using a logger publisher. For more information on configuring event publishers, see [Creating Alerts](#).

1. Log in to the DAS management console using the following URL, if you are not already logged in: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Main**, and then click **Publishers**.
3. Enter the details as shown below to configure the publisher.

Create a New Event Publisher

Enter Event Publisher Details

Event Publisher Name* From Event Source* Stream Attributes	<input type="text" value="product_alerts_publisher"/> <small>⑦ Enter a unique name to identify Event Publisher</small> <input type="text" value="PRODUCT_ALERTS_STREAM:1.0.0"/> <small>⑦ The stream of events that need to be published</small> <small>product-name string, quantity int</small>
To Output Event Adapter Type* Dynamic Adapter Properties Unique Identifier Mapping Configuration Message Format* <small>+ Advanced</small>	
<input type="button" value="Add Event Publisher"/> <input type="button" value="Test Event Publisher"/>	

4. Click **Add Event Publisher**.

Executing the sample

Follow the steps below to execute the sample.

Creating the Spark table which maps to the created event stream

Follow the steps below to create the corresponding virtual table named PRODUCT_ALERTS in the Apache Spark environment within WSO2 DAS to maps with the created event stream PRODUCT_ALERTS_STREAM.

1. Log in to the DAS management console using the following URL, if you are not already logged in: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Main**, and then click **Console**.
3. Enter the following Spark SQL query in the Spark console, and press Enter key to execute it.

```

CREATE TEMPORARY TABLE PRODUCT_ALERTS
USING org.wso2.carbon.analytics.spark.event.EventStreamProvider
OPTIONS (streamName "PRODUCT_ALERTS_STREAM", version "1.0.0",
         payload "product-name STRING, quantity INT"
);

```

Note

The EventStreamProvider works in a way that, it writes event data to a central analytics table, and the event data is picked up by a scheduled task in the receiver nodes, and these events are dispatched to the streams from there. The scheduled task polls the source analytics table in 10 second intervals, to check if there are any event data pending. If required, this scheduled task can be disabled explicitly in a node, by setting the Java system property "disableSparkEventingTask" to "true" (e.g. ./wso2server.sh|bat -DdisableSparkEventingTask=true).

Publishing events to the created event stream

Follow the steps below to publish output events to the created event stream.

1. Log in to the DAS management console using the following URL, if you are not already logged in: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Main**, and then click **Console**.
3. Enter the following Spark SQL query in the Spark console, and press Enter key to execute it.

```

INSERT OVERWRITE TABLE PRODUCT_ALERTS
select * from PRODUCTS_MASTER
where quantity<5;

```

When this query executes, output of the select query is inserted into the PRODUCT_ALERTS table. It reads all the products from the PRODUCTS_STREAM Spark table which have its quantity less than 50. During the query execution, individual rows returned from the select query are published into the PRODUCT_ALERTS stream as events.

1. You view the text events that are published to the CEP server in the logs of it in the CLI as shown below.

```

[2015-07-02 18:34:43,426] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: product_alerts_publisher,
Event: product-name:Adidas,
quantity:3
[2015-07-02 18:34:43,437] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: product_alerts_publisher,
Event: product-name:Puma,
quantity:2
[2015-07-02 18:34:43,437] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: product_alerts_publisher,
Event: product-name:Bettans,
quantity:0

```

Analyzing HTTPD Logs

- [Introduction](#)
- [Prerequisites](#)
- [Building the sample](#)
- [Viewing the output](#)

Introduction

Every time your Web server receives a request, it makes an entry to one or more log files. These log files are useful for a variety of purposes, from statistical analysis of your visitors to forensic analysis of an attack on your server. The HTTPD logs can provide information on everything that happens on your server from the initial request, through the URL mapping process, to the final resolution of the connection (including any errors that may have occurred in the

process). Thereby, HTTPD logs provide you feedback about the activities and performance of the server as well as any problem that may be occurring, to effectively manage your Web server.

This HTTPD logs sample is intended to show the capability of WSO2 DAS which can analyze the raw HTTPD logs and produce useful results. It demonstrates how you can use logs to analyze the Web traffic that comes to your server from different regions. It calculates the region from the IP address in logs and visualizes it through the different mechanisms of presenting data in WSO2 DAS.

Prerequisites

Set up the following prerequisites before you start.

1. Set up the general prerequisites required for WSO2 DAS.
2. Copy the <DAS_HOME>/samples/udfs/org.wso2.das.samples.geoip-3.0.0-SNAPSHOT.jar file (which is the UDF library that is required for this sample), to <DAS_HOME>/repository/components/dropins/ directory.
3. Add the following configurations within the <custom-udf-classes> element of the <DAS_HOME>/repository/conf/analytics/spark/spark-udf-config.xml file.

```
<udf-configuration>
    <custom-udf-classes>
        .....
        <class-name>org.wso2.das.samples.geoip.IPCountryNameUDF</class-name>
        <class-name>org.wso2.das.samples.geoip.IPCountryCodeUDF</class-name>
        .....
    </custom-udf-classes>
</udf-configuration>
```

Building the sample

Follow the steps below to build the sample.

Uploading the Carbon Application

Follow the steps below to upload the Carbon Application (cApp) file of this sample. For more information, see [Carbon Application Deployment 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>/capps/Httpd_Log_Analytics.car file as shown below.

[Home > Manage > Carbon Applications > Add](#)

Add Carbon Applications

Upload Carbon Application

Carbon Application Artifact(.car) Httpd_Log_Analytics.car

4. Click **Main**, then click **Carbon Applications**, and then click **List view**, to see the uploaded Carbon application as shown below.

[Home](#) > [Manage](#) > [Carbon Applications](#) > [List](#)

Carbon Applications List

1 Running Carbon Applications.

Carbon Applications	Version	Actions
Httpd_Log_Analytics	1.0.0	Delete Download

Executing the sample

Follow the steps below to execute the sample.

Running the data publisher

Navigate to `<DAS_HOME>/samples/httpd-logs/` directory in a new CLI tab, and execute the following command to run the data publisher: `ant`

This reads the `<DAS_HOME>/samples/httpd-logs/resources/access.log` file, and sends each log line as an event to the event stream which is deployed through the [above CApp](#).

Viewing the output

You may use the [Data Explorer](#) or the [Analytics Dashboard](#) of the WSO2 DAS Management Console to browse published sample 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 `ORG_WSO2_SAMPLE_HTTPPD_LOGS` for the **Table Name** as shown below.

[Home](#) > [Manage](#) > [Interactive Analytics](#) > [Data Explorer](#)

Data Explorer

Search

Table Name* Schedule Data Purging

Search By Date Range By Query

4. Click **Search**. You view the published HTTPD logs as shown below.

Results							
ORG_WSO2_SAMPLE_HTTPD_LOGS							
	meta_clientType	remotelp	requestDate	request	httpcode	length	_timestamp
external	1.202.218.8	[20/Jun/2012:19:05:12	/robots.txt	404	492	2015-08-19 12:22:55 IST	
external	208.115.113.91	[20/Jun/2012:19:20:16	/logs/?C=M;O=D	200	1278	2015-08-19 12:22:55 IST	
external	123.125.71.20	[20/Jun/2012:19:30:40	/	200	912	2015-08-19 12:22:55 IST	
external	220.181.108.101	[20/Jun/2012:19:31:01	/	200	912	2015-08-19 12:22:55 IST	
external	123.125.68.79	[20/Jun/2012:19:53:24	/	200	625	2015-08-19 12:22:55 IST	
external	178.154.210.252	[20/Jun/2012:19:54:10	?C=S;O=A	200	663	2015-08-19 12:22:55 IST	
external	74.125.126.102	[20/Jun/2012:20:15:28	/	200	606	2015-08-19 12:22:55 IST	
external	74.125.126.103	[20/Jun/2012:20:15:29	/icons/blank.gif	200	383	2015-08-19 12:22:55 IST	
external	74.125.126.93	[20/Jun/2012:20:15:29	/icons/folder.gif	200	460	2015-08-19 12:22:55 IST	
external	74.125.126.82	[20/Jun/2012:20:15:30	/favicon.ico	404	449	2015-08-19 12:22:55 IST	
external	184.82.92.239	[20/Jun/2012:21:03:44	/logs/access.log	200	2519	2015-08-19 12:22:55 IST	
external	173.236.21.106	[20/Jun/2012:21:16:22	/robots.txt	404	488	2015-08-19 12:22:55 IST	
external	173.236.21.106	[20/Jun/2012:21:16:23	/	200	621	2015-08-19 12:22:55 IST	
external	213.186.122.2	[20/Jun/2012:21:27:53	/logs/?C=D;O=D	200	658	2015-08-19 12:22:55 IST	
external	66.249.72.65	[20/Jun/2012:21:28:00	/robots.txt	404	508	2015-08-19 12:22:55 IST	
external	66.249.72.65	[20/Jun/2012:21:28:00	/logs/	200	723	2015-08-19 12:22:55 IST	
external	123.125.71.44	[20/Jun/2012:21:38:57	/	200	913	2015-08-19 12:22:55 IST	
external	220.181.108.88	[20/Jun/2012:21:39:48	/	200	913	2015-08-19 12:22:55 IST	
external	178.154.210.252	[20/Jun/2012:21:45:12	/logs/	200	728	2015-08-19 12:22:55 IST	
external	139.18.2.209	[20/Jun/2012:22:31:43	/	200	912	2015-08-19 12:22:55 IST	
external	123.125.71.48	[20/Jun/2012:22:38:14	/	200	913	2015-08-19 12:22:55 IST	
external	220.181.108.95	[20/Jun/2012:22:39:03	/	200	913	2015-08-19 12:22:55 IST	
external	123.125.67.218	[20/Jun/2012:22:41:50	/robots.txt	404	465	2015-08-19 12:22:55 IST	
external	180.76.6.224	[20/Jun/2012:23:07:26	/	200	908	2015-08-19 12:22:55 IST	
external	208.115.113.91	[20/Jun/2012:23:13:59	/robots.txt	404	469	2015-08-19 12:22:55 IST	

Using the Analytics Dashboard

Follow the steps below to use the [Analytics Dashboard](#) 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 **Analytics Dashboard** in the **Dashboard** menu.
3. Log in to the Analytics Dashboard using admin/admin credentials.
4. You view the HTTPD Log Analysis Dashboard which is already deployed through the CApp as shown below.

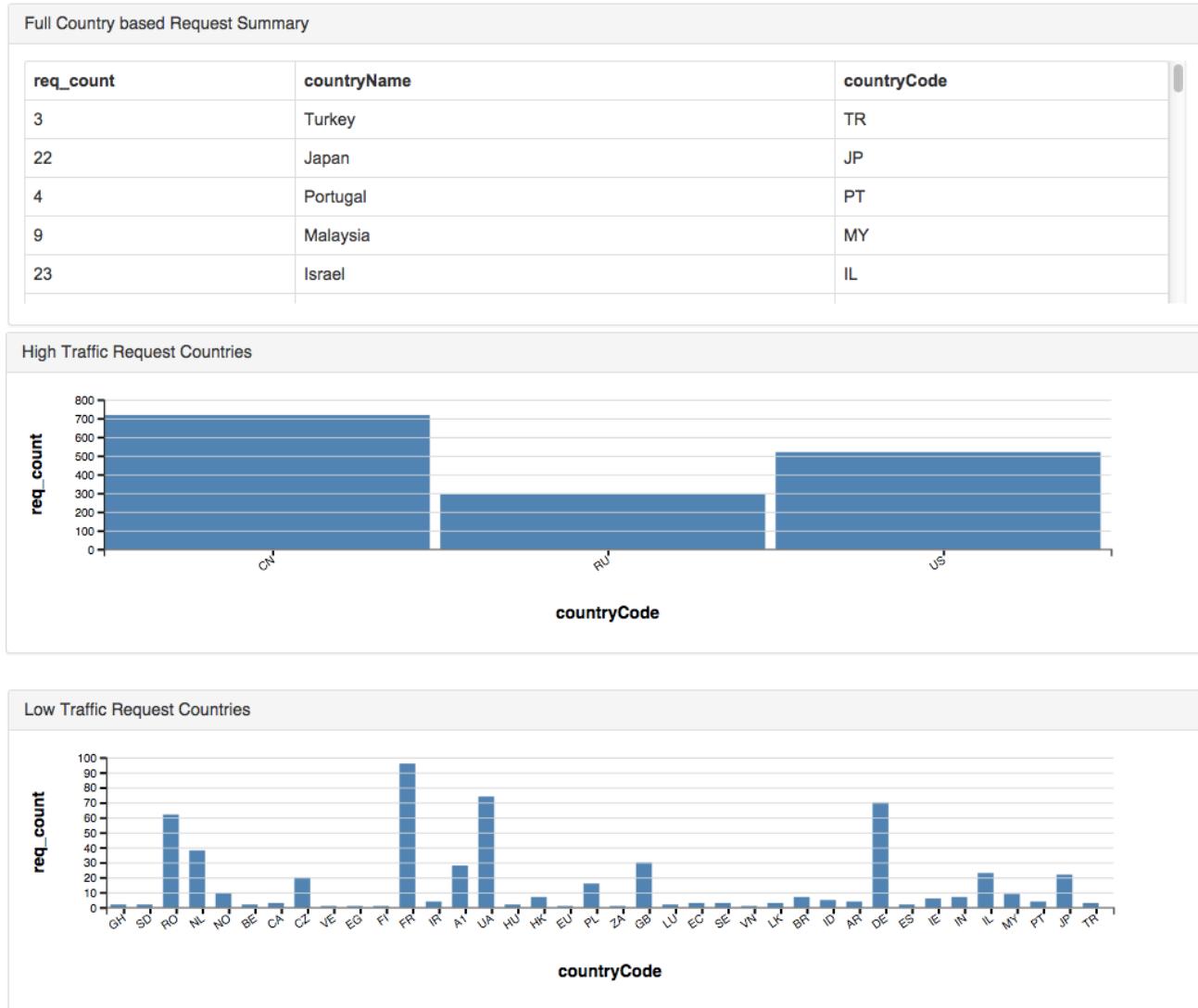


DASHBOARDS

* View, Modify and Delete dashboards



5. Click **View** option of the Dashboard. It opens the HTTPD Log Analysis Dashboard with three gadgets in a new tab of your Web browser as shown below.



Analyzing Smart Home Data

- Introduction
- Prerequisites
- Building the sample
- Viewing the output

Introduction

This sample demonstrates an analysis of data that are collected on the usage of smart home appliances.

Prerequisites

Set up the general prerequisites required for WSO2 DAS, before you start.

Building the sample

Follow the steps below to build the sample.

Uploading the Carbon Application

Follow the steps below to upload the Carbon Application (c-App) file of this sample. For more information, see [Carbo](#)

In Application Deployment 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>/capps/Smart_Home.car` file as shown below.

[Home > Manage > Carbon Applications > Add](#)

Add Carbon Applications

Upload Carbon Application

Carbon Application Artifact(.car) Choose File Smart_Home.car

Upload Cancel

4. Click **Main**, then click **Carbon Applications**, and then click **List view**, to see the uploaded Carbon application as shown below.

[Home > Manage > Carbon Applications > List](#)

Carbon Applications List

1 Running Carbon Applications.

Carbon Applications	Version	Actions
Smart_Home_Sample_CApp	1.0.0	Delete Download

Executing the sample

Follow the steps below to execute the sample.

Running the data publisher

Navigate to `<DAS_HOME>/samples/smart-home/` directory in a new CLI tab, and execute the following command to run the data publisher: `ant`

This reads the `<DAS_HOME>/samples/smart-home/resources/access.log` file, and sends each log line as an event to the event stream which is deployed through the above C-App.

Viewing the output

You may use the **Data Explorer** or the **Analytics Dashboard** of the WSO2 DAS Management Console to browse published sample 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 `ORG_WSO2_SAMPLE_SMART_HOME_DATA` for the **Table Name** as shown below.

Home > Manage > Interactive Analytics > Data Explorer

Data Explorer

Search

Table Name*

Search By Date Range By Query

4. Click **Search**. You view the published data as shown below.

Results							
ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA							
	house_id	metro_area	state	device_id	power_reading	is_peak	_timestamp
17	17	Miami	Florida	1	69.12017	True	2015-09-09 11:08:01 IST
9	9	Chicago	Illinois	2	807.1043	False	2015-09-09 11:08:01 IST
17	17	Dallas	Texas	6	812.3886	True	2015-09-09 11:08:01 IST
15	15	Phoenix	Arizona	6	574.4248	True	2015-09-09 11:08:01 IST
4	4	Seattle	Washington	1	465.85074	False	2015-09-09 11:08:01 IST
18	18	Chicago	Illinois	6	653.92004	False	2015-09-09 11:08:01 IST
4	4	Chicago	Illinois	6	91.548836	True	2015-09-09 11:08:01 IST
13	13	Los Angeles	California	1	122.24013	False	2015-09-09 11:08:01 IST
9	9	Phoenix	Arizona	7	594.87695	True	2015-09-09 11:08:01 IST
21	21	Indianapolis	Indiana	1	255.75494	True	2015-09-09 11:08:01 IST
3	3	Los Angeles	California	7	30.492043	True	2015-09-09 11:08:01 IST
14	14	Phoenix	Arizona	6	677.68317	True	2015-09-09 11:08:01 IST
4	4	San Francisco	California	5	43.83197	False	2015-09-09 11:08:01 IST
6	6	Miami	Florida	1	53.9842	False	2015-09-09 11:08:01 IST
9	9	Miami	Florida	3	365.55484	False	2015-09-09 11:08:01 IST
4	4	Phoenix	Arizona	4	448.37827	False	2015-09-09 11:08:01 IST
12	12	Seattle	Washington	1	19.942368	False	2015-09-09 11:08:01 IST
19	19	Seattle	Washington	3	47.50899	True	2015-09-09 11:08:01 IST
7	7	Phoenix	Arizona	4	95.11217	True	2015-09-09 11:08:01 IST
12	12	Phoenix	Arizona	4	204.07777	False	2015-09-09 11:08:01 IST
3	3	Chicago	Illinois	6	69.09145	False	2015-09-09 11:08:01 IST
3	3	Phoenix	Arizona	5	100.80819	False	2015-09-09 11:08:01 IST
18	18	San Francisco	California	4	333.29944	True	2015-09-09 11:08:01 IST
9	9	San Francisco	California	3	357.27475	True	2015-09-09 11:08:01 IST
15	15	Dallas	Texas	4	268.8887	True	2015-09-09 11:08:01 IST
20	20	San Francisco	California	2	340.40823	True	2015-09-09 11:08:01 IST
2	2	Miami	Florida	3	255.27081	True	2015-09-09 11:08:01 IST
14	14	Miami	Florida	4	582.52515	True	2015-09-09 11:08:01 IST
3	3	Seattle	Washington	7	339.38138	False	2015-09-09 11:08:01 IST
4	4	Phoenix	Arizona	2	230.08377	True	2015-09-09 11:08:01 IST
11	11	Phoenix	Arizona	4	529.74246	False	2015-09-09 11:08:01 IST
21	21	Salt Lake City	Utah	2	37.236732	False	2015-09-09 11:08:01 IST
2	2	Phoenix	Arizona	6	447.7462	False	2015-09-09 11:08:01 IST
14	14	New York	New York	5	36.40989	True	2015-09-09 11:08:01 IST
17	17	Los Angeles	California	1	474.3258	False	2015-09-09 11:08:01 IST
20	20	Seattle	Washington	4	979.823	True	2015-09-09 11:08:01 IST
6	6	Miami	Florida	6	217.11243	True	2015-09-09 11:08:01 IST
13	13	New York	New York	1	81.92053	True	2015-09-09 11:08:01 IST
21	21	New York	New York	6	26.594746	True	2015-09-09 11:08:01 IST
8	8	Salt Lake City	Utah	2	83.217285	True	2015-09-09 11:08:01 IST
3	3	Dallas	Texas	5	40.47768	True	2015-09-09 11:08:01 IST
7	7	Salt Lake City	Utah	4	443.7788	True	2015-09-09 11:08:01 IST
8	8	Dallas	Texas	3	269.855	True	2015-09-09 11:08:01 IST
6	6	Phoenix	Arizona	7	221.43106	False	2015-09-09 11:08:01 IST
18	18	San Francisco	California	1	19.366837	True	2015-09-09 11:08:01 IST
3	3	New York	New York	1	437.72946	True	2015-09-09 11:08:01 IST
5	5	Phoenix	Arizona	4	110.3302	False	2015-09-09 11:08:01 IST
17	17	Seattle	Washington	2	33.70139	False	2015-09-09 11:08:01 IST
4	4	Seattle	Washington	4	220.86511	True	2015-09-09 11:08:01 IST
17	17	Indianapolis	Indiana	4	40.299892	True	2015-09-09 11:08:01 IST

Using the Analytics Dashboard

Follow the steps below to use the [Analytics Dashboard](#) 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 **Analytics Dashboard** in the **Dashboard** menu.
3. Log in to the Analytics Dashboard using admin/admin credentials.

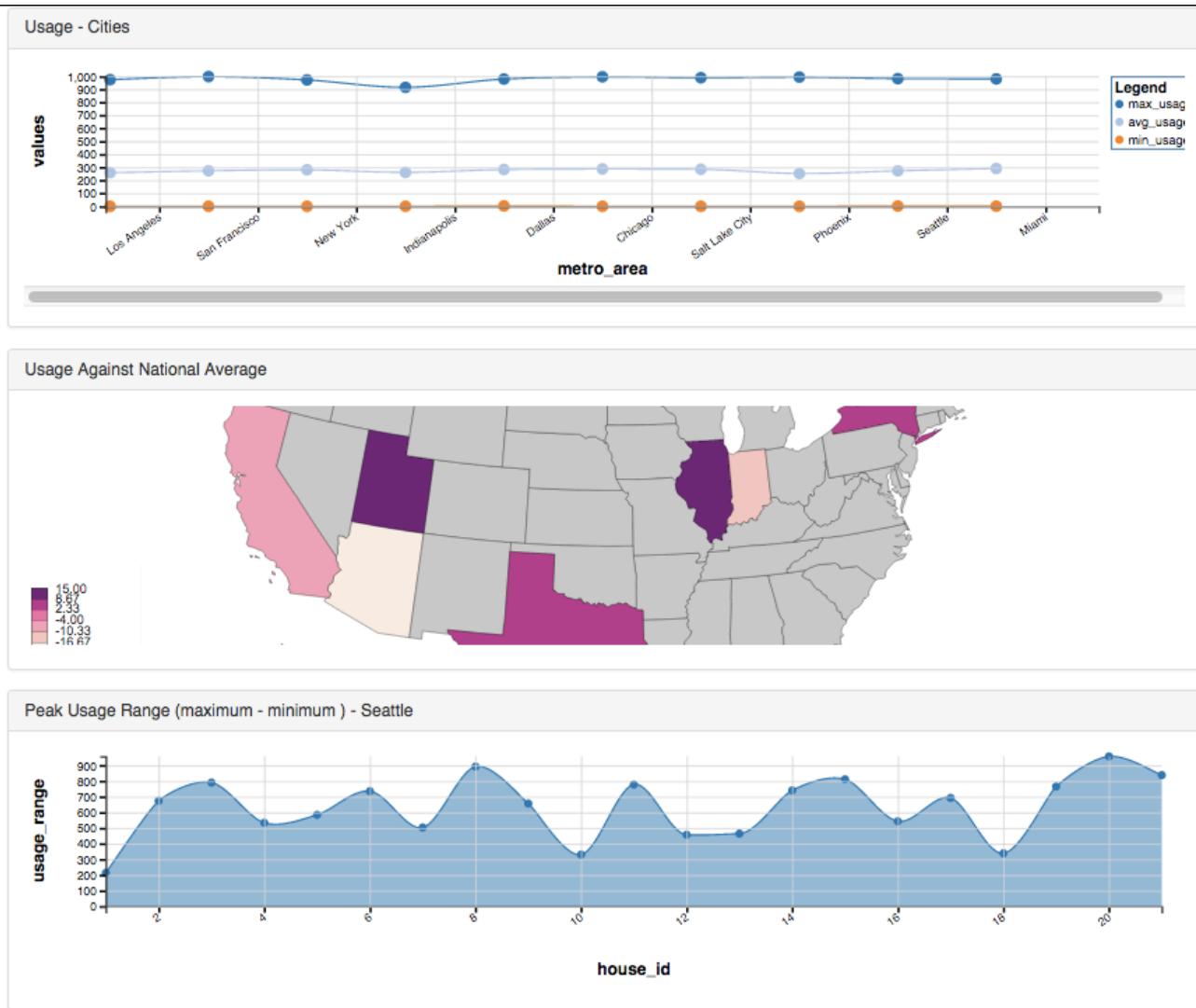
4. You view the Smart Home Dashboard which is already deployed through the C-App as shown below.



DASHBOARDS

* View, Modify and Delete dashboards

5. Click **View** option of the Dashboard. It opens the Smart Home Dashboard with three gadgets in a new tab of your Web browser as shown below.



Analyzing Realtime Service Statistics

- Introduction
- Prerequisites
- Building the sample

- Viewing the output

Introduction

This sample demonstrates an analysis of data collected on the realtime service statistics of a set of requests being sent to the [WSO2 API Manager](#).

Prerequisites

Set up the general prerequisites required for WSO2 DAS, before you start.

Building the sample

Follow the steps below to build the sample.

Uploading the Carbon Application

Follow the steps below to upload the Carbon Application (cApp) file of this sample. For more information, see [Carbon Application Deployment 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>/capps/APIM_Realtime_Analytics.car file as shown below.

[Home](#) > [Manage](#) > [Carbon Applications](#) > [Add](#)

Add Carbon Applications

Upload Carbon Application

Carbon Application Artifact(.car) APIM_Realtime_Analytics.car

4. Click **Main** , then click **Carbon Applications**, and then click **List view** to see the uploaded Carbon application as shown below.

Carbon Applications List

2 Running Carbon Applications.

Carbon Applications	Version	Actions
Smart_Home_Sample_CApp	1.0.0	Delete Download
APIM_Realtime_Analytics	1.0.0	Delete Download

Executing the sample

Follow the steps below to execute the sample.

Running the data publisher

Navigate to <DAS_HOME>/samples/apim-stats/ directory in a new CLI tab, and execute the following command to run the data publisher: ant

This sends randomly generated events to the event stream which is deployed through the above CApp.

Viewing the output

You view the output in the logs of the CLI in which you ran WSO2 DAS as shown below.

```
Buildfile: /home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/apim-stats/build.xml
clean:
[delete] Deleting directory /home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/apim-stats/build

compile:
[mkdir] Created dir: /home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/apim-stats/build/classes
[javac] /home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/apim-stats/build.xml:39: warning: 'includeanruntime' was not set, defaulting to build.sysclasspath=last; set to false for
r repeatable builds
[javac] Compiling 1 source file to /home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/apim-stats/build/classes

jar:
[mkdir] Created dir: /home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/apim-stats/build/jar
[jar] Building jar: /home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/apim-stats/build/jar/HttpdLogs.jar

run:
[echo] Classpath = /home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/axiom-1.2.11.wso2v5.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/commons-logging-1.1.1.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/commons-pool-1.5.6.wso2v1.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/disruptor-2.10.4-wso2v2.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/commons-collect-1.0.0.wso2v2.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/httpclient-4.3.1.wso2v2.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/google-collect-1.0.0.wso2v2.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/org.wso2.carbon.base-4.4.1.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/libthrift-0.7.0.wso2v2.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/org.wso2.carbon.databridge.agent-5.0.1-SNAPSHOT.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/org.wso2.carbon.databridge.common-binary-5.0.1-SNAPSHOT.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/org.wso2.carbon.databridge.common-thrift-5.0.1-SNAPSHOT.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/org.wso2.carbon.databridge.common-logging-4.4.1.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/org.wso2.carbon.utils-4.4.1.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/slf4j-api-1.6.1.jar:/home/maninda/Desktop/das/wso2das-3.0.0-SNAPSHOT/samples/dependencies/slf4j-log4j12-1.6.1.jar
[java] Starting APIM Statistics Agent
[java] Published 10 log events.
[java] log4j:WARN No appenders could be found for logger (org.wso2.carbon.databridge.agent.endpoint.DataEndpoint).
[java] log4j:WARN Please initialize the log4j system properly.
[java] log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
[java] Published 10 log events.
```

Analyzing Wikipedia Data

- [Introduction](#)
- [Prerequisites](#)
- [Building the sample](#)
- [Viewing the output](#)

Introduction

This sample demonstrates an analysis of data that are collected on the usage of the Wikipedia.

Prerequisites

Follow the steps below to set up the prerequisites before you start.

1. Set up the [general prerequisites required for WSO2 DAS](#).
2. Download a [Wikipedia data dump](#), and extract the compressed XML articles dump file to a preferred location of your machine.

Building the sample

Follow the steps below to build the sample.

Uploading the Carbon Application

Follow the steps below to upload the Carbon Application (c-App) file of this sample. For more information, see [Carbon Application Deployment 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>/capps/Wiki[pedia.car` file as shown below.

[Home](#) > [Manage](#) > [Carbon Applications](#) > [Add](#)

Add Carbon Applications

Upload Carbon Application

Carbon Application Artifact(.car) Wikipedia.car

4. Click **Main**, then click **Carbon Applications**, and then click **List view**, to see the uploaded Carbon application as shown below.

[Home](#) > [Manage](#) > [Carbon Applications](#) > [List](#)

Carbon Applications List

1 Running Carbon Applications.

Carbon Applications	Version	Actions
Wikipedia_Sample_CApp	1.0.0	Delete Download

Executing the sample

Follow the steps below to execute the sample.

Tuning the server configurations

The wikipedia dataset is transferred as a single article in a single event. Therefore, an event is relatively large (~300KB). Hence, you need to tune the server configurations as follows.

1. Edit the values of the following properties in the `<DAS_HOME>/repository/conf/data-bridge/data-bridge-config.xml` file as shown below, to tune the queue sizes available for data receiving .

```
<dataBridgeConfiguration>
    <maxEventBufferCapacity>5000</maxEventBufferCapacity>
    <eventBufferSize>2000</eventBufferSize>
</dataBridgeConfiguration>
```

2. Edit the values of the following properties in the `<DAS_HOME>/repository/conf/data-bridge/data-agent-config.xml` file as shown below., to tune the queue sizes available for data persistence .

```
<DataAgentsConfiguration>
    <Agent>
        <Name>Thrift</Name>
        <QueueSize>65536</QueueSize>
        <BatchSize>500</BatchSize>
    </Agent>
</DataAgentsConfiguration>
```

3. Edit the values of the following properties in the `<DAS_HOME>/repository/conf/analytics/analytics-eventsink-config.xml` file as shown below, to change the Thrift publisher related configurations .

```
<AnalyticsEventSinkConfiguration>
  <QueueSize>65536</QueueSize>
  <maxQueueCapacity>1000</maxQueueCapacity>
  <maxBatchSize>1000</maxBatchSize>
</AnalyticsEventSinkConfiguration>
```

Running the data publisher

Navigate to `<DAS_HOME>/samples/wikipedia/` directory in a new CLI tab, and execute the following command to run the data publisher: `ant -Dpath=/home/laf/Downloads/enwiki-20150805-pages-articles.xml -Dcount=1000`

Set the values of the `-D path` and `-Dcount` Java system properties in the above command, to point them to the location where you stored the Wikipedia article XML dump file which you downloaded in [Prerequisites](#), and to the number of articles you need to publish as events out of the total dataset respectively. (E.g. `-Dcount=-1` to publish all articles.) This sends events to the event stream which is deployed through the [sample C-App](#).

Executing the scripts

Follow the steps below to execute the Spark scripts which are deployed by the [sample C-App](#).

1. Log in to the DAS management console using the following URL: https://<DAS_HOST>:<DAS_PORT>/carbon/
2. Click **Main**, and then click **Scripts** in the **Batch Analytics** menu.
3. Click the corresponding **Execute** option of each of the following scripts to execute them.

[Home](#) > [Manage](#) > [Batch Analytics](#) > [Scripts](#)

Available Analytics Scripts

Scripts	Actions
wiki_avg_article_length_script	
wiki_contributor_summary_script	
wiki_total_article_length_script	
wiki_total_article_pages_script	

Viewing the output

You may use the [Data Explorer](#) or the [Analytics Dashboard](#) of the WSO2 DAS Management Console to browse published sample events.

Using the Data Explorer

Follow the steps below to use the [Data Explorer](#) to view the output.

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 `ORG_WSO2_DAS_SAMPLE_WIKIPEDIA_DATA` for the **Table Name** as shown below.

Home > Manage > Interactive Analytics > Data Explorer

Data Explorer

Search

Table Name* **ORG_WSO2_DAS_SAMPLE_WIKIPEDIA_DATA** Schedule Data Purging

Search By Date Range By Query

Search **Reset**

You can also select the other streams which are deployed by the sample C-App as shown below.

Home > Manage > Interactive Analytics > Data Explorer

Data Explorer

Search

Table Name* **ORG_WSO2_DAS_SAMPLE_WIKIPEDIA_DATA** Schedule Data Purging

Search By Date Range By Query

Search **Reset**

Select a Table
 ORG_WSO2_DAS_SAMPLE_WIKIPEDIA_DATA
 WIKI_CONTRIBUTOR_SUMMARY
 WIKI_AVG_ARTICLE_LENGTH
 WIKI_TOTAL_ARTICLE_LENGTH
 WIKI_TOTAL_ARTICLE_PAGES

- Click **Search**. You view the published data as shown below.

Home > Manage > Interactive Analytics > Data Explorer

Help

Data Explorer

Results

ORG_WSO2_DAS_SAMPLE_WIKIPEDIA_DATA

	sha1	title	revision_ts	contributor_username	contributor_id	comment	model	format	text	length	_timestamp
≡	4ro7vvppa5kmm0o1egfjztzcw0vabw	AccessibleComputing	1414279223000	Paine Ellsworth	9092818	add [[WP:RCAT rcat]]s	wikitext	text/x-wiki	#REDIRECT [[Computer accessibility]] {{Redri(move from CamelCase up)}}	69	2015-09-23 14:24:00 IST

<< < 1 > >> Go to page: 1 Row count: 50 Showing 1 of 1

Using the Analytics Dashboard

Follow the steps below to use the **Analytics Dashboard** to view the output.

- Log in to the DAS management console if you are not already logged in.
- Click **Main**, and then click **Analytics Dashboard** in the **Dashboard** menu.
- Log in to the Analytics Dashboard using admin/admin credentials.
- Click the following **CREATE DASHBOARD** button in the top navigational bar to create a new dashboard.



- Enter a **Title** and a **Description** for the new dashboard as shown below, and click **Next** as shown below.

CREATE A DASHBOARD

Name of your Dashboard

Wikipedia Samples Dashboard

URL

wikipedia-samples-dashboard

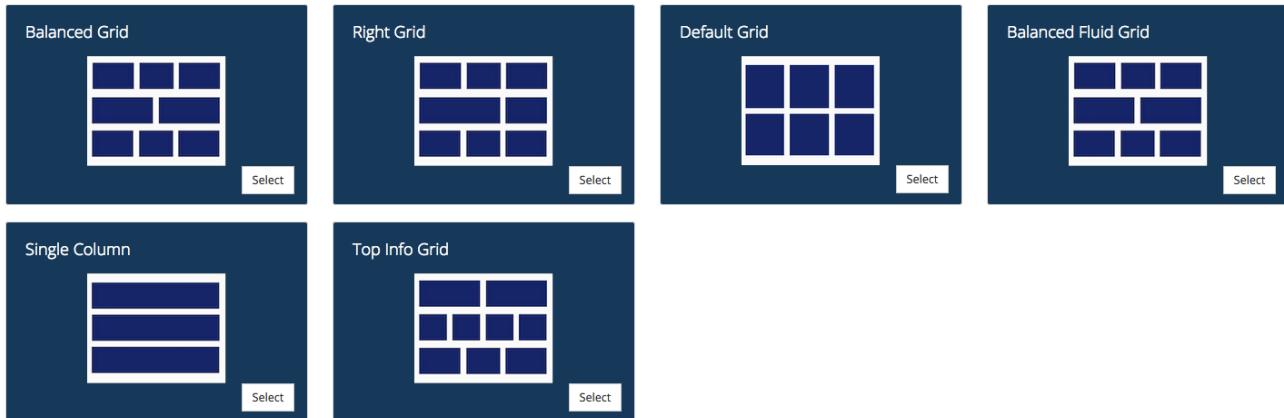
Description

E.g. Monthly Sales Statistics

[Next](#)

6. Select a layout to place its components as shown below.

PLEASE SELECT A LAYOUT



7. Click **Select** button of the **Single Column** layout. You view a layout editor with the chosen layout blocks marked using dashed lines.
8. Click the following **CREATE GADGET** button in the top menu bar.

CREATE GADGET

9. Select the input data source as shown below, and click **Next**.

CREATE A GADGET

[Select Table/Event Stream](#)

[Configure Chart](#)

Analytics Table/Event Stream

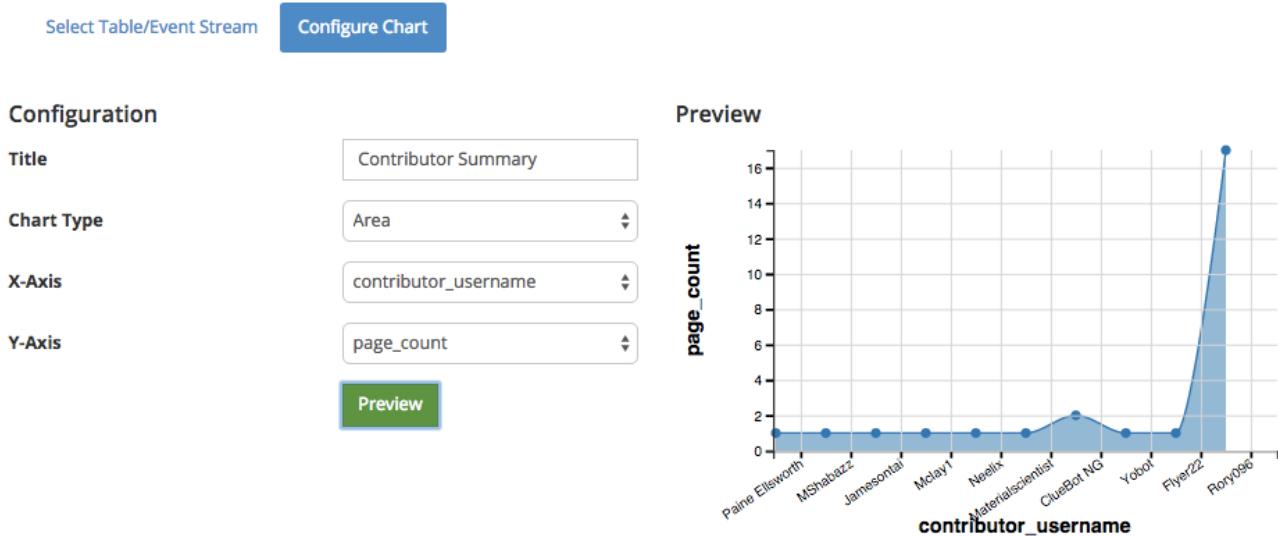
WIKI_CONTRIBUTOR_SUMMARY [batch]

Fields

- contributor_username
- page_count

10. Select **Chart Type** and enter the preferred x, y axis and additional parameters based on the selected chart type as shown below, and click **Preview**.

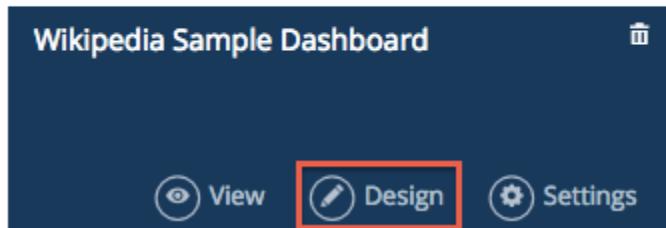
CREATE A GADGET



11. Click **Add to Gadget Store**.
 12. Click the corresponding **Design** button of the Wikipedia_Samples_Dashboard to add the Contributor Summary gadget as shown below.

DASHBOARDS

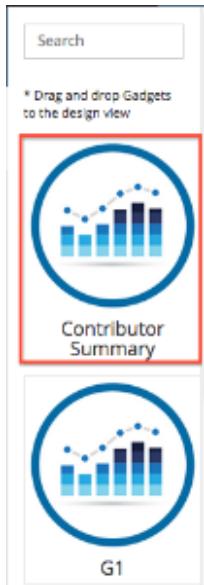
* View, Modify and Delete dashboards



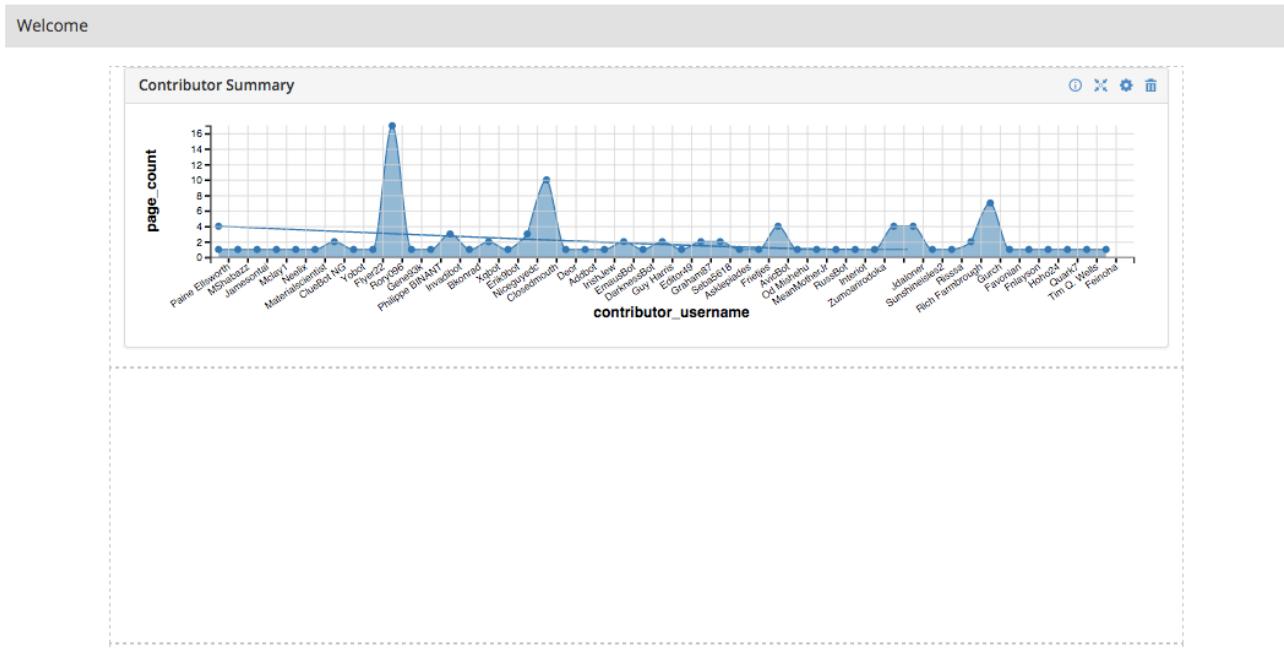
13. Click the following gadget browser icon in the side menu bar.



You view the new gadget listed in the gadget browser as shown below.



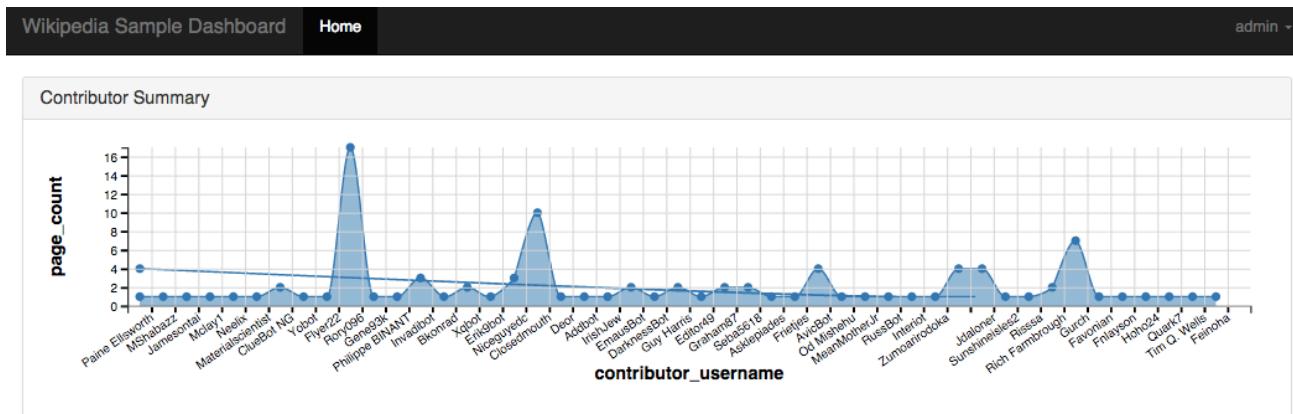
14. Click on the new gadget, drag it out, and place it in the preferred grid of the selected layout in the dashboard editor as shown below.



15. Click the following **PREVIEW** button in the top menu bar.



You view the preview of the `wikipedia_Samples_Dashboard` with the Contributor Summary gadget added to it as shown below.



Reference Guide

The following topics provide reference information for working with WSO2 DAS:

- REST APIs for Analytics Data Service
- Default Ports of WSO2 Products
- WSO2 Patch Application Process
- Calling Admin Services from Apps
- Customizing the Management Console
- Using Analytics Spark Features in Other WSO2 Products
- WSO2 DAS Performance Analysis
- Working with Dashboards

REST APIs for Analytics Data Service

This feature exposes the interface of the analytics data service as REST APIs.

- [Analytics REST API Guide](#)
- [CORS Settings for the Analytics REST API](#)
- [HTTP Status Codes](#)

Analytics REST API Guide

The following are the REST APIs that are implemented in WSO2 DAS 3.1.0.

Function	REST API
Checking if a table exists	GET /analytics/table_exists?table={tableName}
Listing all tables	GET /analytics/tables
Retrieving records of a table	GET /analytics/tables/{tableName}/{from}/{to}/{start}/{count}
Getting the record count of a table	GET /analytics/tables/{tableName}/recordcount
Searching records of a table	POST /analytics/search
Defining/updating the schema of a table	POST /analytics/tables/{tableName}/schema
Getting the search record count of a table	POST /analytics/search_count
Timing out the indexing process completion	GET /analytics/indexing_done?timeout=<long-value>
Getting the schema of a table	GET /analytics/tables/{tableName}/schema
Drilling down through hierarchical categories	POST /analytics/facets
Retrieving matching records through a drill down search	POST /analytics/drilldown
Retrieving the count of matching records through a drill down search	POST /analytics/drillDownScoreCount
Retrieving records using given primary key value pairs	POST /analytics/tables/{tableName}/keyed_records
Retrieving All Record Stores	GET /analytics/recordstores
Retrieving the Record Store of a Given Table	GET /analytics/recordstore?table={tableName}

Checking if the Given Records Store Supports Pagination	GET /analytics/pagination/<recordstore-name>
Tracking the Indexing Process Completion of a Table	GET /analytics/indexing_done?maxWait=10&table={tableName}
Retrieving Aggregated Values of Given Records	POST /analytics/aggregates
Retrieving the Event Count of Range Facets	POST /analytics/rangecount
Re-indexing the Records of a Given Table	POST /analytics/tables/{tableName}/{from}/{to}
Clear Index Information of a Given Table via REST API	DELETE /analytics/tables/{tableName}/indexData

The REST APIs are secured with basic authentication. Therefore, follow the steps below to add a basic auth header when calling these methods.

1. Build a string of the form `username:password`.
 2. Encode the string you created above using Base64. For encoding the above string using Base64, see [Encode to Base64 format](#).
 3. Define an authorization header with the term "Basic_ ", followed by the encoded string. For example, the basic auth authorization header using "admin" as both username and password is as follows:
- Authorization: Basic YWRtaW46YWRtaW4=

Cross-Origin Resource Sharing (CORS) should be enabled if you are using analytics REST API from outside the DAS domain, or if the REST API caller is situated in a machine with a different host/port configuration to WSO2 DAS. For more information, see [CORS Settings for the Analytics REST API](#).

Checking if a Given Table Exists via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Check if a particular table exists.
Resource Path	/analytics/table_exists?table={tableName}
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic
Username	admin
Password	admin

The user name and the password (i.e. admin:admin) should be encoded into the Base64 format and included in the GET request.

Parameter description

Parameter	Description
{tableName}	The name of the table that you want to find in the DAS server.

Sample cURL command

```
c u r l           - k           - X
GET 'https://localhost:9443/analytics/table_exists?table=<tableName>' -H "Authorization: Basic bsaee64(username:password)"
```

Example

```
curl -k -XGET 'https://localhost:9443/analytics/table_exists?table=table1' -H
"Authorization: Basic YWRtaW46YWRtaW4="
```

Sample output

If the table that you searched for exists, an output similar to the following is received.

```
{
  "status": "success",
  "message": "Table : table1 exists."
}
```

If the table that you searched for does not exist, an output similar to the following is received.

```
{
  "status": "non-existent",
  "message": "Table : table1 does not exist."
}
```

REST API response

HTTP status code	200 or 404
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Retrieving the List of Tables of All Record Stores via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Listing all tables
Resource Path	/analytics/tables
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Sample cURL command

```
curl -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v
https://localhost:9443/analytics/tables -k
```

Example

```
GET https://localhost:9443/analytics/tables
```

Sample output

```
[ "TABLE1", "TABLE2", "TABLE3" ]
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving Records Based on a Time Range via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Retrieving records of a table
Resource Path	/analytics/tables/{tableName}/{from}/{to}/{start}/{count}
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	The name of the table from which the records are retrieved.
{from}	The starting time to retrieve records from. This is an optional parameter.
{to}	The ending time up to which records should be retrieved. This is an optional parameter.
{start}	The paginated index from value. This is an optional parameter.
{count}	The paginated records count to be read. This is an optional parameter.

Sample cURL command

```
curl -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v
https://localhost:9443/analytics/tables/ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA/1451969331
166/1451969331167/0/3 -k
```

Example

```
GET https://localhost:9443/analytics/tables/testtable/1421314718339/1421919518339
```

Sample output

```
[
{
  "id": "258d8bac-bba7-5e59-d63a-322b4dfcb620",
  "tableName": "ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA",
  "timestamp": 145196933166,
  "values": { "_version": "1.0.0", "is_peak": true, "state": "Florida", "metro_area": "Miami", "device_id": 4, "power_reading": 484.4254, "house_id": 19 }
},
{
  "id": "fddcdedb7-fdf7-a92d-4f7b-a630e2a0db49",
  "tableName": "ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA",
  "timestamp": 1451969331166,
  "values": { "_version": "1.0.0", "is_peak": false, "state": "Arizona", "metro_area": "Phoenix", "device_id": 1, "power_reading": 92.64153, "house_id": 13 }
},
{
  "id": "65a81fdc-08bf-a8f5-4128-8fc59861ed25",
  "tableName": "ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA",
  "timestamp": 1451969331166,
  "values": { "_version": "1.0.0", "is_peak": false, "state": "Texas", "metro_area": "Dallas", "device_id": 3, "power_reading": 259.73157, "house_id": 2 }
}
]
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving the Total Record Count of a Table via REST API

- Overview
- Prerequisites
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

This will return a value for the record count only if the underlying [Record Store](#) has record count support.

Description	Getting the record count of a table
Resource Path	/analytics/tables/{tableName}/recordcount
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	Name of the table of which, the record count is required

Prerequisites

Before you retrieve the total record count of a table, the following should be included in the relevant database configuration in the `DAS_HOME/repository/conf/analytics/rdbms-config.xml` file.

```
<recordCountSupported>true</recordCountSupported>
```

Sample cURL command

```
curl -H "Authorization: Basic YWRtaW46YWRtaW4=" -v
https://localhost:9443/analytics/tables/ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA/recordcount -k
```

Example

```
GET https://localhost:9443/analytics/tables/testtable/recordcount
```

Sample output

```
3000
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving All Records Matching the Given Search Query via REST API

- Overview
- Prerequisites
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Searching records of a table
Resource Path	POST
HTTP Method	/analytics/search
Request/Response Format	application/json
Authentication	Basic

Prerequisites

The records in the event table need to be indexed in order to be retrieved. e.g., The state attribute (in the sample cURL command below) should be indexed before publishing data to the [event stream](#) in which it is included. For more information on indexing, see [Configuring Indexes](#). Additionally it supports sorting by an indexed field. Records can be sorted by multiple fields. If two records has the same value, then it will sort those two records by the second field in the given array in "sortBy" attribute.

Sample cURL command

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/search -d '{"tableName": "ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA", "query": "state:Texas", "start": 0, "count": 3, "sortBy": [{"field": "device_id", "sortType": "ASC"}]}' -k
```

Example

```
POST https://localhost:9443/analytics/search
{
  "tableName": "books",
  "query": "name:MyBook",
  "start": 0,
  "count": 100,
  "sortBy": [
    {
      "field": "PRICE",
      "sortType": "ASC"
    },
    {
      "field": "SIZE",
      "sortType": "ASC"
    }
  ]
}
```

Sample output

```
[
  {
    "id": "14242656601230.5739142271249453",
    "tableName": "Books",
    "timestamp": 1424265660123,
    "values": {
      "PRICE": 100,
      "SIZE": 200,
      "version": 1,
      "name": "MyBook"
    }
  },
  {
    "id": "14242656601230.5739142271249454",
    "tableName": "Books",
    "timestamp": 1424265660124,
    "values": {
      "PRICE": 120,
      "SIZE": 120,
      "version": 2,
      "name": "MyBook"
    }
  }
]
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving the Number of Records Matching the Given Search Query via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Getting the search record count of a table
Resource Path	POST
HTTP Method	/analytics/search_count
Request/Response Format	application/json
Authentication	Basic

Sample cURL command

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/search_count -d '{"tableName": "ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA", "query": "state:Texas", "start": 0, "count": 3}' -k
```

Example

```
POST https://localhost:9443/analytics/search_count
{
  "tableName": "testtable",
  "query": "product:wso2cdm"
}
```

Sample output

```
10
```

REST API response

HTTP status code	200
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Tracking the Indexing Process Completion via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Waiting for the timeout of the indexing process completion
Resource Path	/analytics/indexing_done?timeout=<long-value>
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{timeout}	The timeout in seconds after waiting until the indexing process completes. If the indexing process is not complete before the elapse of the number of seconds specified, no status message relating to the indexing process completion is returned. However, the indexing process continues until completion.

Sample cURL command

```
curl -H "Content-Type: application/json" -H "Authorization: Basic
base64(username:password)" -v
https://localhost:9443/analytics/indexing_done?table=<table-name> -k
```

Example

```
GET https://localhost:9443/analytics/indexing_done?table=testTable
```

Sample output

```
{
    status: "success"
    message: "Indexing completed successfully"
}
```

REST API response

HTTP status code	200 or 500
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Clear Index Information of a Given Table via REST API

- [Overview](#)
- [Sample cURL command](#)
- [Example](#)
- [Sample output](#)

- REST API response

Overview

Description	Clearing all the index information of a given table.
Resource Path	/analytics/tables/{tableName}/indexData
HTTP Method	DELETE
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	The name of the table with the index information to be deleted.

Sample cURL command

```
curl -X DELETE -H "Content-Type: application/json" -H "Authorization: Basic
YWRtaW46YWRtaW4=" -v
"https://localhost:9443/analytics/tables/ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA/indexData"
-k
```

Example

```
DELETE https://localhost:9443/analytics/tables/testtable/indexData
```

Sample output

```
{ "status" : "success", "message" : "Successfully cleared indices in table: testtable"
}
```

REST API response

HTTP status code	200
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Defining/Updating the Schema of a Table

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Defining the schema of a table, or updating the existing schema of a table.
Resource Path	/analytics/tables/{tableName}/schema
HTTP Method	POST
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	The name of the table for which the schema is defined.

Sample cURL command

```
POST https://localhost:9443/analytics/tables/testtable/schema{
  "columns": {
    "house_id": {
      "type": "INTEGER",
      "isScoreParam": false,
      "isIndex": false
    },
    "metro_area": {
      "type": "STRING",
      "isScoreParam": false,
      "isIndex": false
    },
    "state": {
      "type": "STRING",
      "isScoreParam": false,
      "isIndex": false
    },
    "device_id": {
      "type": "INTEGER",
      "isScoreParam": false,
      "isIndex": false
    },
    "power_reading": {
      "type": "FLOAT",
      "isScoreParam": false,
      "isIndex": false
    },
    "is_peak": {
      "type": "BOOLEAN",
      "isScoreParam": false,
      "isIndex": false
    }
  },
  "primaryKeys": [
    "house_id",
    "device_id"
  ]
}
```

Example

```
https://localhost:9443/analytics/tables/testtable/schema
```

Sample output

```
{
    "status": "success",
    "message": "Successfully set table schema for table:testtable"
}
```

REST API response

HTTP status code	200, 404
------------------	----------

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving the Schema of a Table via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Getting the schema of a table
Resource Path	/analytics/tables/{tableName}/schema
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	The name of the table of which the schema is required.

Sample cURL command

```
curl -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v
https://localhost:9443/analytics/tables/ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA/schema -k
```

Example

```
GET https://localhost:9443/analytics/tables/testtable/schema
```

Sample output

```
{
  "columns": [
    {"house_id": {"type": "INTEGER", "isScoreParam": false, "isIndex": false}, "metro_area": {"type": "STRING", "isScoreParam": false, "isIndex": false}, "state": {"type": "STRING", "isScoreParam": false, "isIndex": false}, "device_id": {"type": "INTEGER", "isScoreParam": false, "isIndex": false}, "power_reading": {"type": "FLOAT", "isScoreParam": false, "isIndex": false}, "is_peak": {"type": "BOOLEAN", "isScoreParam": false, "isIndex": false}},
    "primaryKeys": []
  }
```

REST API response

HTTP status code	200, 404
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Drilling Down Through Categories via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Drilling down through hierarchical categories
Resource Path	/analytics/facets
HTTP Method	POST
Request/Response Format	application/json
Authentication	Basic

Sample cURL command

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/facets -d '{"tableName": "sampleTableName", "fieldName": "sampleFieldName", "categoryPath": ["parent", "child", "grandChild"], "query": "timestamp : [1213343534535 TO 465464564644]", "scoreFunction": "weight*popularity"}' -k
```

Example

```
POST https://localhost:9443/analytics/facets
{
    "tableName" : "sampleTableName",
    "fieldName" : "sampleFieldName",
    "categoryPath" : ["parent", "child", "grandChild"],
    "query" : "timestamp : [1213343534535 TO 465464564644]",
    "scoreFunction" : "weight*popularity"
}
```

Sample output

```
{
    "categoryPath" : ["parent", "child", "grandChild"],
    "categories" : {grandGrandChild1 : 15, grandGrandChild2 : 25, grandGrandChild3 : 42}
}
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving Specific Records through a Drill Down Search via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Retrieving matching records through a drill-down search. In addition to that, records are sorted by the field and the sortType given in the "sortBy" attribute.
Resource Path	/analytics/drilldown
HTTP Method	POST
Request/Response Format	application/json
Authentication	Basic

Sample cURL command

```
curl -X POST -H "Content-type: application/json" -H 'Authorization: Basic YWRtaW46YWRtaW4=' -v https://localhost:9443/analytics/drilldown -d '{
    "tableName": "testtable",
    "categories": [
        {
            "fieldName": "location",
            "path" : ["Sri lanka", "Colombo"]
        },
        {
            "fieldName": "publishedDate",
            "path" : ["2015", "Mar", "23"]
        }
    ],
    "query" : "timestamp : [1243214324532 TO 4654365223]",
    "recordStart" : 0,
    "recordCount" : 100,
    "sortBy" : [
        {
            "field" : "_timestamp",
            "sortType" : "DESC"
        }
    ]
}' -k
```

Example

```
POST https://localhost:9443/analytics/drilldown

{
    "tableName": "testtable",
    "categories": [
        {
            "fieldName": "location",
            "path" : ["Sri lanka", "Colombo"]
        },
        {
            "fieldName": "publishedDate",
            "path" : ["2015", "Mar", "23"]
        }
    ],
    "query" : "timestamp : [1243214324532 TO 4654365223]",
    "recordStart" : 0,
    "recordCount" : 100,
    "sortBy" : [
        {
            "field" : "_timestamp",
            "sortType" : "DESC"
        }
    ]
}
```

Sample output

```
[
  {
    "id": "14242656601230.5739142271249453",
    "tableName": "testtable",
    "timestamp": 1424265660123,
    "values": {
      "product": "wso2cdm",
      "location": ["Srilanka", "Colombo"],
      "publishedData": ["2015", "Mar", "23"]
    }
  },
  {
    "id": "14242656601230.5739142271249451",
    "tableName": "testtable",
    "timestamp": 1424265660103,
    "values": {
      "product": "wso2cdm",
      "location": ["Srilanka", "Colombo"],
      "publishedData": ["2015", "Mar", "23"]
    }
  }
]
```

REST API response

HTTP status code	200
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Retrieving the Number of Records Matching the Drill Down Criteria via REST API

- [Overview](#)
- [Sample cURL command](#)
- [Example](#)
- [Sample output](#)
- [REST API response](#)

Overview

Description	Retrieving the count of matching records through a drill-down search
Resource Path	/analytics/drillDownScoreCount
HTTP Method	POST
Request/Response Format	application/json
Authentication	Basic

Sample cURL command

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/drilldownScoreCount -d '{
    "tableName": "<table-Name>",
    "categories": [
        {
            "fieldName": "<field-Name>",
            "path" : [ "<Parent-category>" , "<Child-Categoey>" ]
        },
        {
            "fieldName": "<field-Name>",
            "path" : [ "<Parent-category>" , "<Child-category>" ,
            "<child-child-category>" ]
        }
    ],
    "query" : "<lucene-query>"
}' -k
```

Example

```
POST https://localhost:9443/analytics/drillDownScoreCount

{
    "tableName": "testtable",
    "categories": [
        {
            "fieldName": "location",
            "path" : [ "Sri lanka" , "Colombo" ]
        },
        {
            "fieldName": "publishedDate",
            "path" : [ "2015" , "Mar" , "23" ]
        }
    ],
    "query" : "timestamp : [1243214324532 TO 4654365223]"
}
```

Sample output

3

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving Records Matching the Given Primary Key Combination via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Retrieve records of a table using given primary key value pairs
Resource Path	/analytics/tables/{tableName}/keyed_records
HTTP Method	POST
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	The name of the table of which records are being retrieved using primary key value pairs.

Sample cURL command

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/tables/<table-Name>/keyed_records -d '{
"valueBatches" : [
    "<primaryKey>" : "<primaryKey-Value>",
    "<primaryKey>" : "<primaryKey-Value>",
    "<primaryKey>" : "<primaryKey-Value>"
],
{
    "<primaryKey>" : "<primaryKey-Value>",
    "<primaryKey>" : "<primaryKey-Value>",
}
],
"columns" : [ "<column1>" , "<column2>" , "<column3>" ]
}' -k
```

response

```
[
    {
        "id": "record-id",
        "tableName": "<table-name>",
        "timestamp": "<time-stamp>",
        "values": {
            "<field1>" : "<value1>",
            "<field2>" : "<value2>",
            "<field3>" : "<value3>"
        }
    }
    ...
]
```

Example

```
POST https://localhost:9443/analytics/tables/testtable/keyed_records
{
"valueBatches" : [
    "key1" : "value1",
    "key2" : "value2",
    "key3" : "value3"
},
{
    "key4" : "value4",
    "key5" : "value5"
],
"columns" : [ "column1" , "column2" , "column3" ]
}
```

Sample output

```
[
  [
    {
      "id": "14242656601230.5739142271249453",
      "tableName": "table2",
      "timestamp": 1424265660123,
      "values": {
        "key1": "value1",
        "key2": "value2",
        "key3": "value3"
      }
    }
  ]
  [
    [
      {
        "id": "14242656601230.5739142271249454",
        "tableName": "table2",
        "timestamp": 1424265660123,
        "values": {
          "key1": "value1",
          "key2": "value2",
          "key3": "value3"
        }
      }
    ]
  ]
  [
    [
      {
        "id": "14242656601230.5739142271249455",
        "tableName": "table2",
        "timestamp": 1424265660123,
        "values": {
          "key4": "value4",
          "key5": "value5"
        }
      }
    ]
  ]
]
```

REST API response

HTTP status code	200 or 404
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Retrieving All Record Stores via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Retrieving all record stores
Resource Path	/analytics/recordstores

HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Sample cURL command

```
curl -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/recordstores -k
```

Example

```
GET https://localhost:9443/analytics/recordstores
```

Sample output

```
[ "EVENT_STORE" , "PROCESSED_DATA_STORE" , "INDEX_STAGING_STORE" ]
```

REST API response

HTTP status code	200
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Retrieving the Record Store of a Given Table via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Retrieving the record store of a given table
Resource Path	/analytics/recordstore?table={tableName}
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	The name of the table of which the record store is retrieved

Sample cURL command

```
curl -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/recordstore?table=ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA -k
```

Example

```
GET https://localhost:9443/analytics/recordstore?table=testTable
```

Sample output

```
EVENT_STORE
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Checking if the Given Records Store Supports Pagination via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Checking if the given record store supports pagination
Resource Path	/analytics/pagination/{recordstore-name}
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{recordstore-name}	The name of the record store for which you need to carry out a check as to whether it supports pagination or not.

Sample cURL command

```
curl -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v
https://localhost:9443/analytics/pagination/EVENT_STORE -k
```

Example

```
GET https://localhost:9443/analytics/pagination/EVENT_STORE
```

Sample output

```
{"status": "success", "message": "RecordStore : EVENT_STORE support pagination."}
```

REST API response

HTTP status code	200
For descriptions of the HTTP status codes, see HTTP Status Codes .	

Tracking the Indexing Process Completion of a Table via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Tracking if the indexing process is complete of a given table.
Resource Path	/analytics/indexing_done?table={tableName}
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	The name of the table for which you need to check whether the indexing process is complete.

Sample cURL command

```
curl -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/indexing_done?table=ORG_WSO2_DAS_SAMPLE_SMART_HOME_DA TA -k
```

Example

```
GET https://localhost:9443/analytics/indexing_done?table=testTable
```

Sample output

```
{
    status: "success"
    message: "Indexing completed successfully"
}
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving Aggregated Values of Given Records via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Retrieving aggregated values of given records
Resource Path	/analytics/aggregates
HTTP Method	POST
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
tableName	The name of the event table for which this query should be run.

parentPath	This specifies whether the aggregate should be calculated for a child category within a specific category. When you specify a parent path, everything that is not included in that path is ignored when the <code>groupBy</code> operation is executed. Therefore, this parameter also plays the role of a filter.
aggregateLevel	This specifies the level up to which the aggregation should be done. This is specified relative to the parent path.
groupByField	This represents the <code>GROUP BY</code> equivalent in SQL, and specifies the field by which the retrieved values should be grouped.
aggregateFields	<p>This is an array of JSON objects. Each object represents the field that you want to aggregate.</p> <ul style="list-style-type: none"> • <code>fieldName</code>: This is the field that should be aggregated. • <code>aggregate</code>: This is the aggregate function that should be used. • <code>alias</code>: This is the field name that is used for the aggregated result.

Sample cURL command

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/aggregates -d '{
    "tableName": "<table-name>",
    "parentPath": [<category-path>],
    "aggregateLevel": <aggregate-level>
    "groupByField": "<field-by which we aggregate>",
    "aggregateFields": [
        {
            "fieldName": "<field-name>",
            "aggregate": "<aggregate-Type>",
            "alias": "<alias-for-result>"
        },
        {
            "fieldName": "<field-name>",
            "aggregate": "<aggregate-Type>",
            "alias": "<alias-for-result>"
        }
    ]
}'
```

Example

```
GET https://localhost:9443/analytics/aggregates
{
  "tableName": "test2",
  "parentPath": [],
  "aggregateLevel": 0
  "groupByField": "facet",
  "aggregateFields": [
    {
      "fieldName": "n",
      "aggregate": "AVG",
      "alias": "result_avg"
    },
    {
      "fieldName": "n",
      "aggregate": "MAX",
      "alias": "result_max"
    }
  ]
}
```

Sample output

```
[ {
  "id": "14399961246350.13125980939205978",
  "tableName": "test2", "timestamp": 1439996124610,
  "values": {
    "result_avg": "20",
    "result_max": "22",
    "single_valued_facet_field": [engineering] //This should be a facet with a single value.
  }
},
{
  "id": "14399961246350.13125980939205999",
  "tableName": "test2", "timestamp": 1439996124610,
  "values": {
    "result_avg": "34",
    "result_max": "46",
    "single_valued_facet_field": [marketing] //This should be a facet with a single value.
  }
}
]
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

Retrieving the Event Count of Range Facets via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Retrieving the event count of a range facet
Resource Path	/analytics/rangecount
HTTP Method	POST
Request/Response Format	application/json
Authentication	Basic

Sample cURL command

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic
YWRtaW46YWRtaW4=" -v https://localhost:9443/analytics/rangecount -d '{
    "tableName": "<tableName>",
    "rangeField": "<range-field>",
    "ranges": [
        {
            "label": "<label-given-to-idenitfy-value>",
            "from": "<lower-boundry-inclusive>",
            "to": "<upper-boundry-exclusive>"
        },
        {
            "label": "<label-given-to-idenitfy-value>",
            "from": "<lower-boundry-inclusive>",
            "to": "<upper-boundry-exclusive>"
        }
    ],
    "query": "<lucene-query>"
}' -k
```

Example

```
POST https://localhost:9443/analytics/rangecount
{
    "tableName": "testtable",
    "rangeField": "distance",
    "ranges": [
        {
            "label": "1km - 3km",
            "from": 1000,
            "to": 3000
        },
        {
            "label": "3km - 6km",
            "from": 3000,
            "to": 6000
        }
    ],
    "query": "location": "NYC"
}
```

Sample output

```
1km - 3km: 8
3km - 6km: 6
```

REST API response

HTTP status code	200
	For descriptions of the HTTP status codes, see HTTP Status Codes .

Re-indexing the Records of a Given Table via REST API

- Overview
- Sample cURL command
- Example
- Sample output
- REST API response

Overview

Description	Re-index the records of a given table.
Resource Path	/analytics/tables/{tableName} ?from={from}&to={to}
HTTP Method	POST
Request/Response format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	The name of the table with the records to be re-indexed.
{from}	The starting time of the time period to which the records to be indexed belong.
{to}	The ending time of the time period to which the records to be indexed belong.

Sample cURL command

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -v "https://localhost:9443/analytics/tables/ORG_WSO2_DAS_SAMPLE_SMART_HOME_DATA?from=1451969331166&to=1451969331167" -k
```

Example

```
POST  
https://localhost:9443/analytics/tables/testtable?from=1421314718339&to=1421919518339
```

Sample output

```
{ "status" : "success", "message" : "re-indexing..." }
```

REST API response

HTTP status code	200
------------------	-----

For descriptions of the HTTP status codes, see [HTTP Status Codes](#).

CORS Settings for the Analytics REST API

Cross-Origin Resource Sharing (CORS) is a mechanism used by client-side processes to access resources from domains outside their own. This allows such processes to overcome the standard same-origin policy, which prohibits access to external resources/APIs. To use the [analytics REST API](#) from outside WSO2 DAS domain, or if the REST API caller is situated in a machine with a different host/port configuration to WSO2 DAS, you need to enable CORS for the analytics REST API.

Follow the steps below to enable CORS for the analytics REST API.

1. Navigate to <DAS_HOME>/repository/deployment/server/webapps/analytics/WEB-INF/web.xml file.
2. Add the following configuration within the <web-app> element.

```

<filter>
    <filter-name>CorsFilter</filter-name>
    <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
    <init-param>
        <param-name>cors.allowed.origins</param-name>
        <param-value>{domains to be allowed, comma-separated}</param-value>
    </init-param>
    <init-param>
        <param-name>cors.allowed.methods</param-name>
        <param-value>GET, POST, HEAD, OPTIONS, PUT, DELETE, PATCH</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CorsFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

3. Add the domains that you intend to access the REST API as a comma-separated list, within the `<param-value>` element under the parameter name `cors.allowed.origins`.

Allowing CORS for the REST API allows access to all the domains specified under the parameter name `cors.allowed.origins`. Therefore, list only the required domains as values for this parameter to minimize possible security issues.

The analytics REST API uses the in-built CORS filter of Apache Tomcat to achieve this functionality. For all available parameters that could be specified for this filter, see [Container Provided Filters](#).

HTTP Status Codes

When REST API requests are sent to carryout various actions, various HTTP status codes will be returned based on the state of the action (success or failure) and the HTTP method (POST, GET, PUT, DELETE) executed. The following are the definitions of the various HTTP status codes that are returned.

- Success HTTP status codes
- Error HTTP status codes

Success HTTP status codes

Code	Code Summary	Description
200	Ok	HTTP request was successful. The output corresponding to the HTTP request will be returned. Generally used as a response to a successful GET and PUT REST API HTTP methods.
201	Created	HTTP request was successfully processed and a new resource was created. Generally used as a response to a successful POST REST API HTTP method.
204	No content	HTTP request was successfully processed. No content will be returned. Generally used as a response to a successful DELETE REST API HTTP method.
202	Accepted	HTTP request was accepted for processing, but the processing has not been completed. This generally occurs when your successful in trying to undeploy an application.

Error HTTP status codes

Code	Code Summary	Description
404	Not found	Requested resource not found. Generally used as a response for unsuccessful GET and PUT REST API HTTP methods.
409	Conflict	Request could not be processed because of conflict in the request. This generally occurs when you are trying to add a resource that already exists. For example, when trying to add an auto-scaling policy that has an already existing ID.
500	Internal server error	Server error occurred.

Default Ports of WSO2 Products

Following are the default ports that are used for each WSO2 product when the [port offset](#) is 0.

If you are running multiple WSO2 products on the same server, you must set the <offset> value in <PROD UCT_HOME>/repository/conf/carbon.xml to a different value for each product so that there are no port conflicts. For example, if you are running two WSO2 products on the same server, and you set the port offset to 1 in one product and 2 in the second product, the management console port will be changed from the default of 9443 to 9444 in the first product and to 9445 in the second product. See [here](#) for more information on changing the offset.

- Common ports
- Product-specific ports

Common ports

The following ports are common to all WSO2 products that provide the given feature. Some features are bundled in the WSO2 Carbon platform itself and therefore are available in all WSO2 products by default.

Management console ports

WSO2 products that provide a management console use the following servlet transport ports:

- 9443 - HTTPS servlet transport (the default URL of the management console is <https://localhost:9443/carbon>)
- 9763 - HTTP servlet transport

LDAP server ports

Provided by default in the WSO2 Carbon platform.

- 10389 - Used in WSO2 products that provide an embedded LDAP server

KDC ports

- 8000 - Used to expose the Kerberos key distribution center server

JMX monitoring ports

WSO2 Carbon platform uses TCP ports to monitor a running Carbon instance using a JMX client such as JConsole. By default, JMX is enabled in all products. You can disable it using <PRODUCT_HOME>/repository/conf/etc/jmx.xml file.

- 11111 - RMIServer port. Used to monitor Carbon remotely
- 9999 - RMIServer port. Used along with the RMIREGISTRY port when Carbon is monitored from a JMX client that is behind a firewall

Clustering ports

To cluster any running Carbon instance, either one of the following ports must be opened.

- 45564 - Opened if the membership scheme is multicast
- 4000 - Opened if the membership scheme is wka

Random ports

Certain ports are randomly opened during server startup. This is due to specific properties and configurations that become effective when the product is started. Note that the IDs of these random ports will change every time the server is started.

- A random TCP port will open at server startup because of the `-Dcom.sun.management.jmxremote` property set in the server startup script. This property is used for the JMX monitoring facility in JVM.
- A random UDP port is opened at server startup due to the log4j appender (`SyslogAppender`), which is configured in the `<PRODUCT_HOME>/repository/conf/log4j.properties` file.

Product-specific ports

Some products open additional ports.

[API Manager](#) | [BAM](#) | [BPS](#) | [Data Analytics Server](#) | [Complex Event Processor](#) | [Elastic Load Balancer](#) | [ESB](#) | [Identity Server](#) | [Message Broker](#) | [Machine Learner](#) | [Storage Server](#) | [Enterprise Mobility Manager](#)

API Manager

- 10397 - Thrift client and server ports
- 8280, 8243 - NIO/PT transport ports
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to BAM/CEP: stat pub

If you change the default API Manager ports with a port offset, most of its ports will be changed automatically according to the offset except a few exceptions described in the [APIM Manager documentation](#).

BAM

- 9160 - Cassandra port using which Thrift listens to clients
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to BAM
- 7611 - Thrift TCP port to receive events from clients to BAM
- 21000 - Hive Thrift server starts on this port

BPS

- 2199 - RMI registry port (datasources provider port)

Data Analytics Server

- 9160 - Cassandra port on which Thrift listens to clients
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to DAS
- 7611 - Thrift TCP port to receive events from clients to DAS
- For a list of Apache Spark related ports, see [WSO2 Data Analytics Server Documentation - Spark Configurationss](#).

Complex Event Processor

- 9160 - Cassandra port on which Thrift listens to clients
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to CEP
- 7611 - Thrift TCP port to receive events from clients to CEP
- 11224 - Thrift TCP port for HA management of CEP

Elastic Load Balancer

- 8280, 8243 - NIO/PT transport ports

ESB

Non-blocking HTTP/S transport ports: Used to accept message mediation requests. If you want to send a request to an API or a proxy service for example, you must use these ports. ESB_HOME}/repository/conf/axis2/axis2.xml file.

- 8243 - Passthrough or NIO HTTPS transport
- 8280 - Passthrough or NIO HTTP transport

Identity Server

- 8000 - KDCServerPort. Port which KDC (Kerberos Key Distribution Center) server runs
- 10500 - ThriftEntitlementReceivePort

Message Broker

Message Broker uses the following JMS ports to communicate with external clients over the JMS transport.

- 5672 - Port for listening for messages on TCP when the AMQP transport is used.
- 8672 - Port for listening for messages on TCP/SSL when the AMQP Transport is used.
- 1883 - Port for listening for messages on TCP when the MQTT transport is used.
- 8833 - Port for listening for messages on TCP/SSL when the MQTT Transport is used.
- 7611 - The port for Apache Thrift Server.

Machine Learner

- 7077 - The default port for Apache Spark.
- 54321 - The default port for H2O.
- 4040 - The default port for Spark UI.

Storage Server

Cassandra:

- 7000 - For Inter node communication within cluster nodes
- 7001 - For inter node communication within cluster nodes via SSL
- 9160 - For Thrift client connections
- 7199 - For JMX

HDFS:

- 54310 - Port used to connect to the default file system.
- 54311 - Port used by the MapRed job tracker
- 50470 - Name node secure HTTP server port
- 50475 - Data node secure HTTP server port
- 50010 - Data node server port for data transferring
- 50075 - Data node HTTP server port
- 50020 - Data node IPC server port

Enterprise Mobility Manager

The following ports need to be opened for Android and iOS devices so that it can connect to Google Cloud Messaging (GCM)/Firebase Cloud Messaging (FCM) and APNS (Apple Push Notification Service) and enroll to WSO2 EMM.

Android:

The ports to open are 5228, 5229 and 5230. GCM/FCM typically only uses 5228, but it sometimes uses 5229 and 5230.

GCM/FCM does not provide specific IPs, so it is recommended to allow the firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169.

iOS:

- 5223 - TCP port used by devices to communicate to APNs servers
- 2195 - TCP port used to send notifications to APNs
- 2196 - TCP port used by the APNs feedback service
- 443 - TCP port used as a fallback on Wi-Fi, only when devices are unable to communicate to APNs on port 5223

The APNs servers use load balancing. The devices will not always connect to the same public IP address for notifications. The entire 17.0.0.0/8 address block is assigned to Apple, so it is best to allow this range in the firewall settings.

API Manager:

The following WSO2 API Manager ports are only applicable to WSO2 EMM 1.1.0 onwards.

- 10397 - Thrift client and server ports
- 8280, 8243 - NIO/PT transport ports

Changing the offset for default ports

When you run multiple WSO2 products, multiple instances of the same product, or multiple WSO2 product clusters on the same server or virtual machines (VMs), you must change their default ports with an offset value to avoid port conflicts. The default HTTP and HTTPS ports (without offset) of a WSO2 product are 9763 and 9443 respectively. Port offset defines the number by which all ports defined in the runtime such as the HTTP/S ports will be changed. For example, if the default HTTP port is 9763 and the port offset is 1, the effective HTTP port will change to 9764. For each additional WSO2 product instance, you set the port offset to a unique value. The default port offset is 0.

There are two ways to set an offset to a port:

- Pass the port offset to the server during startup. The following command starts the server with the default port incremented by 3: `./wso2server.sh -DportOffset=3`
- Set the Ports section of `<PRODUCT_HOME>/repository/conf/carbon.xml` as follows: `<Offset>3</Offset>`

Usually, when you offset the port of the server, all ports it uses are changed automatically. However, there are few exceptions as follows in which you have to change the ports manually according to the offset. The following table indicates the changes that occur when the offset value is modified.

WSO2 Server instance	PortOffset	Sample Default Port Value
WSO2 Product 1	0	9443
WSO2 Product 2	1	9444
WSO2 Product 3	2	9445

WSO2 Product 4	3	9446
WSO2 Product 5	4	9447

WSO2 DAS Specific Ports

Following are the ports which WSO2 DAS specifically uses.

All the below port numbers are subject to any [port offset](#) you applied.

Ports inherited from WSO2 BAM

WSO2 DAS inherits the following port configurations used in its predecessor, [WSO2 Business Activity Monitor \(BAM\)](#).

- 7711 - Thrift SSL port for secure transport, where the client is authenticated to use WSO2 DAS.
- 7611 - Thrift TCP port where WSO2 DAS receives events from clients.

For a complete set of information on port configurations of WSO2 DAS and related WSO2 products, see [Default Ports of WSO2 Products](#).

Ports used by the Spark Analytics Engine

The Spark Analytics engine is used in 3 separate modes in WSO2 DAS as follows.

- Local mode
- Cluster mode
- Client mode

Default port configurations for these modes, are as follows.

For more information on these ports, go to [Apache Spark Documentation](#).

Ports available for all modes

The ports that are available for all the above 3 modes are listed below.

Description	Port number
spark.ui.port	4040
spark.history.ui.port	18080
spark.blockManager.port	12000
spark.broadcast.port	12500
spark.driver.port	13000
spark.executor.port	13500
spark.filesServer.port	14000
spark.replicaClassServer.port	14500

Ports available for the cluster mode

The ports that are available only for the cluster mode are listed below.

Description	Port number
spark.master.port	7077
spark.master.rest.port	6066
spark.master.webui.port	8081
spark.worker.port	11000
spark.worker.webui.port	11500

WSO2 Patch Application Process

You apply patches to WSO2 products either as individual patches or through a service pack. A service pack is recommended when the number of patches increase. The following sections explain the WSO2 patch application process:

- Applying service packs to the Kernel
- Applying individual patches to the Kernel
- Verifying the patch application
- Overview of the patch application process

Before you begin

- You can download all WSO2 Carbon Kernel patches from [here](#).
- Before you apply a patch, check its `README.txt` file for any configuration changes required.

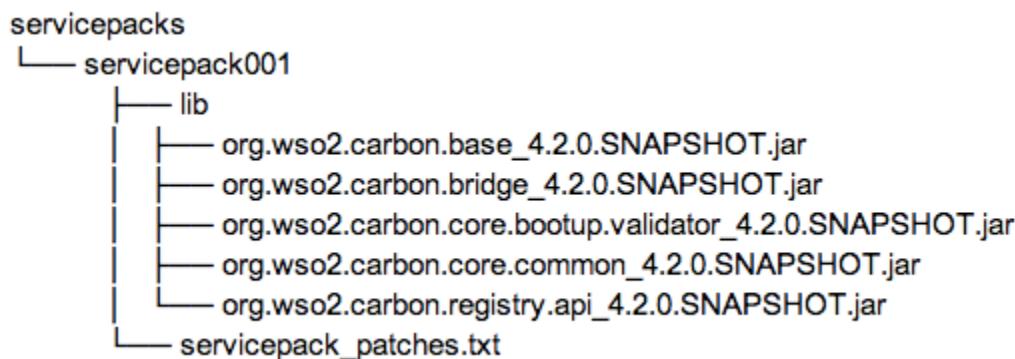
Applying service packs to the product

Carbon 4.2.0 Kernel supports service packs. A service pack is a collection of patches in a single pack. It contains two elements:

- The `lib` directory: contains all the JARs relevant to the service pack.
- The `servicepack_patches.txt` text file: contains the list of JARs in the service pack.

Follow the steps below to apply service packs to your product.

1. Copy the service pack file to the `<PRODUCT_HOME>/repository/components/servicepacks/` directory. For example, the image below shows how a new service pack named `servicepack001` is added to this directory.



2. Start your product. The following steps will be executed:

- a. Before applying any patches, the process first creates a backup folder named patch0000 inside the <PRODUCT_HOME>/repository/components/patches/ directory, which will contain the original content of the <PRODUCT_HOME>/repository/components/plugins/ directory. This step enables you to revert back to the previous state if something goes wrong during operations.
- b. The latest service pack in the <PRODUCT_HOME>/repository/components/servicepacks/ directory will be applied. That is, the patches in the service pack will be applied to the <PRODUCT_HOME>/repository/components/plugins/ directory.
- c. In addition to the service pack, if there are individual patches added to the <PRODUCT_HOME>/repository/components/patches/ directory, those will also be incrementally applied to the plugins directory.

The metadata file available in the service pack will maintain a list of the applied patches by service pack. Therefore, the metadata file information will be compared against the <PRODUCT_HOME>/repository/components/patches/ directory, and only the patches that were not applied by the service pack will be incrementally applied to the plugins directory.

Applying individual patches to the product

You can apply each patch individually to your system as explained below. Alternatively, you can [apply patches through service packs](#) as explained above.

1. Copy the patches to the <PRODUCT_HOME>/repository/components/patches/ directory.
2. Start the Carbon server. The patches will then be incrementally applied to the plugins directory.

Before applying any patches, the process first creates a backup folder named patch0000 inside the <PRODUCT_HOME>/repository/components/patches/ directory, which will contain the original content of the <PRODUCT_HOME>/repository/components/plugins/ directory. This step enables you to revert back to the previous state if something goes wrong during operations.

Prior to Carbon 4.2.0, users were expected to apply patches by starting the server with `wso2server.sh -DapplyPatches`. Now, you do not have to issue a special command to trigger the patch application process. It starts automatically if there are changes in either the <PRODUCT_HOME>/repository/components/servicepacks/ directory or the <PRODUCT_HOME>/repository/components/patches/ directory. It verifies all the latest JARs in the servicepacks and patches directories against the JARs in the plugins directory by comparing MD5s of JARs.

Verifying the patch application

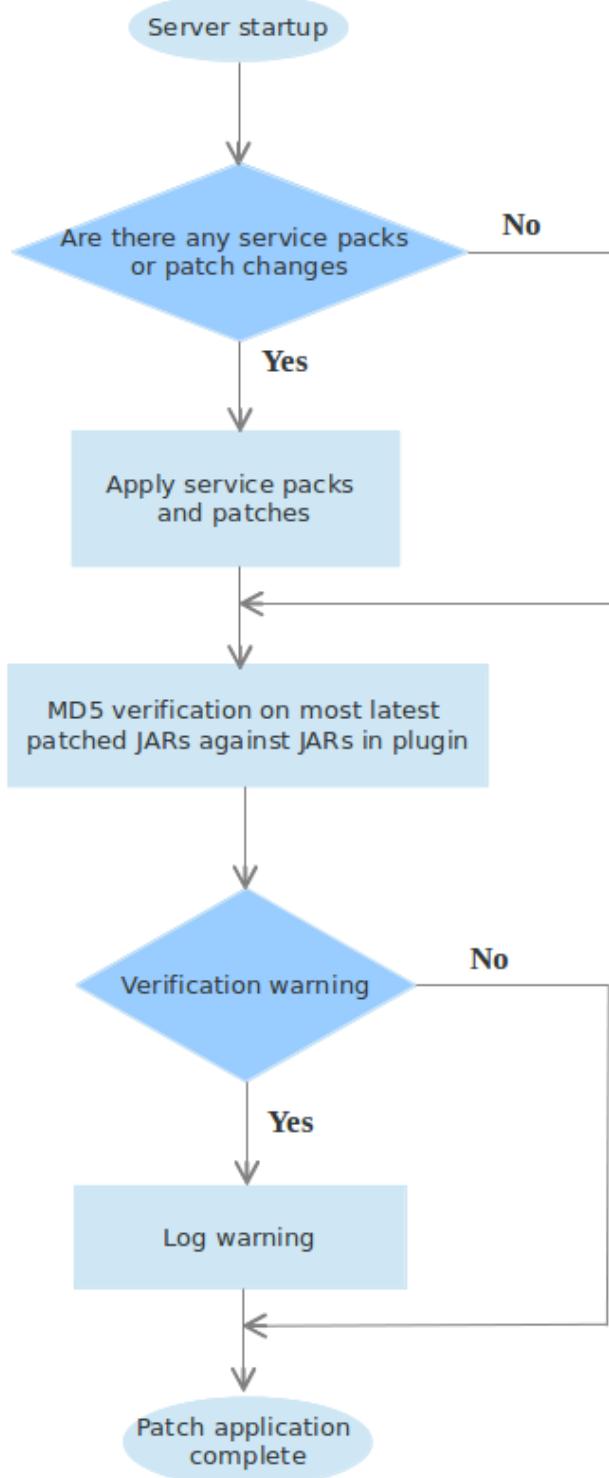
After the patch application process is completed, the patch verification process ensures that the latest service pack and other existing patches are correctly applied to the <PRODUCT_HOME>/repository/components/plugins/ folder.

- All patch related logs are recorded in the <PRODUCT_HOME>/repository/logs/patches.log file.
- The <PRODUCT_HOME>/repository/components/patches/.metadata/prePatchedJARs.txt meta file contains the list of patched JARs and the md5 values.
- A list of all the applied service packs and patches are in the <PRODUCT_HOME>/repository/components/default/configuration/prePatched.txt file.

Do not change the data in the <PRODUCT_HOME>/repository/components/default/configuration/prePatched.txt file. The patch application process gets the pre-patched list from this file and compares the list with the patches available in the servicepack and patches directories. If you change the data in this file, you will get a startup error when applying patches.

Overview of the patch application process

The diagram below shows how the patch application process is implemented when you start the server.



Calling Admin Services from Apps

WSO2 products are managed internally using SOAP Web services known as **admin services**. WSO2 products come with a management console UI, which communicates with these admin services to facilitate administration capabilities through the UI.

A service in WSO2 products is defined by the following components:

- Service component: provides the actual service
- UI component: provides the Web user interface to the service
- Service stub: provides the interface to invoke the service generated from the service WSDL

There can be instances where you want to call back-end Web services directly. For example, in test automation, to minimize the overhead of having to change automation scripts whenever a UI change happens, developers prefer to call the underlying services in scripts. The topics below explain how to discover and invoke these services from your applications.

Discovering the admin services

By default, the WSDLs of admin services are hidden from consumers. Given below is how to discover them.

1. Set the `<HideAdminServiceWSDLs>` element to false in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file.
2. Restart the server.
3. Start the WSO2 product with the `-DosgiConsole` option, such as `sh <PRODUCT_HOME>/bin/wso2server.sh -DosgiConsole` in Linux.
4. When the server is started, hit the enter/return key several times to get the OSGI shell in the console.
5. In the OSGI shell, type: `osgi> listAdminServices`

6. The list of admin services of your product are listed. For example:

```
osgi> listAdminServices
Admin services deployed on this server:
1. ProvisioningAdminService, ProvisioningAdminService, https://192.168.219.1:8243/services/ProvisioningAdminService
2. SynapseApplicationAdmin, SynapseApplicationAdmin, https://192.168.219.1:8243/services/SynapseApplicationAdmin
3. CarbonAppUploader, CarbonAppUploader, https://192.168.219.1:8243/services/CarbonAppUploader
4. OperationAdmin, OperationAdmin, https://192.168.219.1:8243/services/OperationAdmin
5. SequenceAdminService, SequenceAdminService, https://192.168.219.1:8243/services/SequenceAdminService
6. MediationLibraryAdminService, MediationLibraryAdminService, https://192.168.219.1:8243/services/MediationLibraryAdminService
7. StatisticsAdmin, StatisticsAdmin, https://192.168.219.1:8243/services/StatisticsAdmin
8. LoggedUserInfoAdmin, LoggedUserInfoAdmin, https://192.168.219.1:8243/services/LoggedUserInfoAdmin
9. MediationStatisticsAdmin, MediationStatisticsAdmin, https://192.168.219.1:8243/services/MediationStatisticsAdmin
10. TopicManagerAdminService, TopicManagerAdminService, https://192.168.219.1:8243/services/TopicManagerAdminService
11. MessageProcessorAdminService, MessageProcessorAdminService, https://192.168.219.1:8243/services/MessageProcessorAdminService
12. ApplicationAdmin, ApplicationAdmin, https://192.168.219.1:8243/services/ApplicationAdmin
13. NDataSourceAdmin, NDataSourceAdmin, https://192.168.219.1:8243/services/NDataSourceAdmin
14. ServiceGroupAdmin, ServiceGroupAdmin, https://192.168.219.1:8243/services/ServiceGroupAdmin
15. ClassMediatorAdmin, ClassMediatorAdmin, https://192.168.219.1:8243/services/ClassMediatorAdmin
```

7. To see the service contract of an admin service, select the admin service's URL and then paste it in your browser with **?wsdl** at the end. For example: <https://localhost:9443/services/UserAdmin?wsdl>

In products like WSO2 ESB and WSO2 API Manager, the port is 8243 (assuming 0 port offset). However, you should be accessing the Admin Services via the management console port, which is 9443 when there is no port offset.

8. Note that the admin service's URL appears as follows in the list you discovered in step 6:

```
AuthenticationAdmin, AuthenticationAdmin, https://<host  
IP>:8243/services/AuthenticationAdmin
```

Invoking an admin service

Admin services are secured using common types of security protocols such as HTTP basic authentication, WS-Security username token, and session based authentication to prevent anonymous invocations. For example, the UserAdmin Web service is secured with the HTTP basic authentication. To invoke a service, you do the following:

1. Authenticate yourself and get the session cookie.
2. Generate the client stubs to access the back-end Web services.

To generate the stubs, you can write your own client program using the Axis2 client API or use an existing tool like [SoapUI](#) (4.5.1 or later) or wsdl2java.

The wsdl2java tool, which comes with WSO2 products by default hides all the complexity and presents you with a proxy to the back-end service. The stub generation happens during the project build process within the Maven POM files. It uses the Maven ant run plug-in to execute the wsdl2java tool.

You can also use the Java client program given [here](#) to invoke admin services. All dependency JAR files that you need to run this client are found in the /lib directory.

Authenticate the user

The example code below authenticates the user and gets the session cookie:

```

import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;
import org.wso2.carbon.authenticator.stub.AuthenticationAdminStub;
import org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;
import org.apache.axis2.context.ServiceContext;
import java.rmi.RemoteException;

public class LoginAdminServiceClient {
    private final String serviceName = "AuthenticationAdmin";
    private AuthenticationAdminStub authenticationAdminStub;
    private String endPoint;

    public LoginAdminServiceClient(String backEndUrl) throws AxisFault {
        this.endPoint = backEndUrl + "/services/" + serviceName;
        authenticationAdminStub = new AuthenticationAdminStub(endPoint);
    }

    public String authenticate(String userName, String password) throws
RemoteException,
                                            LoginAuthenticationExceptionException {

        String sessionCookie = null;

        if (authenticationAdminStub.login(userName, password, "localhost")) {
            System.out.println("Login Successful");

            ServiceContext serviceContext = authenticationAdminStub.
                _getServiceClient().getLastOperationContext().getServiceContext();
            sessionCookie = (String)
serviceContext.getProperty(HTTPConstants.COOKIE_STRING);
            System.out.println(sessionCookie);
        }

        return sessionCookie;
    }

    public void logOut() throws RemoteException,
LogoutAuthenticationExceptionException {
        authenticationAdminStub.logout();
    }
}

```

To resolve dependency issues, if any, add the following dependency JARs location to the class path: <PRODUCT_HOME>/repository/components/plugins.

The the AuthenticationAdminStub class requires org.apache.axis2.context.ConfigurationContext as a parameter. You can give a null value there.

Generate the client stubs

After authenticating the user, give the retrieved admin cookie with the service endpoint URL as shown in the sample below. The service management service name is ServiceAdmin. You can find its URL (e.g., <https://localhost:9443/services/ServiceAdmin>) in the service.xml file in the META-INF folder in the respective bundle that you find in <PRODUCT_HOME>/repository/components/plugins.

```

import org.apache.axis2.AxisFault;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.wso2.carbon.service.mgt.stub.ServiceAdminStub;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaDataWrapper;
import java.rmi.RemoteException;

public class ServiceAdminClient {
    private final String serviceName = "ServiceAdmin";
    private ServiceAdminStub serviceAdminStub;
    private String endPoint;

    public ServiceAdminClient(String backEndUrl, String sessionCookie) throws AxisFault
    {
        this.endPoint = backEndUrl + "/services/" + serviceName;
        serviceAdminStub = new ServiceAdminStub(endPoint);
        //Authenticate Your stub from sessionCookie
        ServiceClient serviceClient;
        Options option;

        serviceClient = serviceAdminStub._getServiceClient();
        option = serviceClient.getOptions();
        option.setManageSession(true);
        option.setProperty(org.apache.axis2.transport.http.HTTPConstants.COOKIE_STRING,
sessionCookie);
    }

    public void deleteService(String[] serviceGroup) throws RemoteException {
        serviceAdminStub.deleteServiceGroups(serviceGroup);
    }

    public ServiceMetaDataWrapper listServices() throws RemoteException {
        return serviceAdminStub.listServices("ALL", "*", 0);
    }
}

```

The following sample code lists the back-end Web services:

```

import org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaData;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaDataWrapper;

import java.rmi.RemoteException;

public class ListServices {
    public static void main(String[] args)
        throws RemoteException, LoginAuthenticationExceptionException,
               LogoutAuthenticationExceptionException {
        System.setProperty("javax.net.ssl.trustStore",
                           "$ESB_HOME/repository/resources/security/wso2carbon.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", "wso2carbon");
        System.setProperty("javax.net.ssl.trustStoreType", "JKS");
        String backEndUrl = "https://localhost:9443";

        LoginAdminServiceClient login = new LoginAdminServiceClient(backEndUrl);
        String session = login.authenticate("admin", "admin");
        ServiceAdminClient serviceAdminClient = new ServiceAdminClient(backEndUrl,
session);
        ServiceMetaDataWrapper serviceList = serviceAdminClient.listServices();
        System.out.println("Service Names:");
        for (ServiceMetaData serviceData : serviceList.getServices()) {
            System.out.println(serviceData.getName());
        }

        login.logOut();
    }
}

```

Customizing the Management Console

The Management Console user interface (<https://localhost:9443/carbon>) of a Carbon product consists of two layers:

1. **UI inherited from WSO2 Carbon platform** contains the templates, styles (css files), and images that are stored in the core Carbon UI bundle stored in <PRODUCT_HOME>/repository/components/plugins/org.wso2.carbon.ui_<version-number>.jar where <version-number> is the version of the Carbon kernel that the product is built on. This bundle is responsible for the overall look and feel of the entire Carbon platform.
2. **UI unique to each product** contains all the styles and images that override the ones in core Carbon platform. This file is in <PRODUCT_HOME>/repository/components/plugins/org.wso2.<product-name>.styles_<version-number>.jar where <version-number> is the version of the product.

The following topics explain how to download a Carbon product and customize its user interface.

- Setting up the development environment
- Customizing the user interface
- Starting the server

Setting up the development environment

To download and set up the product environment for editing, take the following steps.

1. Download your product.
2. Extract the ZIP file into a separate folder in your hard drive.
3. Go to the <PRODUCT_HOME>/repository/components/plugins/ directory to find the required JAR files:
 - org.wso2.carbon.ui_<version-number>.jar
 - org.wso2.<product-name>.styles_<version-number>.jar
4. Copy the JAR files to a separate location on your hard drive. Since the JAR files are zipped, you must unzip them to make them editable.

You can now customize the look and feel of your product by modifying the contents of the JAR files as described in the next section.

Customizing the user interface

Customizing the product interface involves changing the layout/design of the Carbon framework as well as changing the styles and images specific to the product. The following topics explain how some of the main changes to the product interface can be done.

- Changing the layout
- Changing the styles on the Carbon framework
- Changing the product specific styles and images

Changing the layout

The layout of the Carbon framework is built using a tiles JSP tag library. The use of tiles allows us to break the presentation of the layout into small JSP snippets that perform a specific function. For example, header.jsp and footer.jsp are the tiles corresponding to the header and footer in the layout. The template.jsp file controls the main layout page of the Carbon framework, which holds all the tiles together. That is, the header part in the template.jsp file is replaced with the <tiles:insertAttribute name="header" /> tag, which refers to the header.jsp file. The template.jsp file as well as the JSP files corresponding to the tiles are located in the org.wso2.<product-name>.styles_<version-name>.jar/web/admin/layout/ directory.

Therefore, changing the layout of your product primarily involves changing the template.jsp page (main layout page) and the JSP files of the relevant JSP tiles.

Ensure that you do not change or remove the ID attributes on the .jsp files.

Changing the styles on the Carbon framework

The global.css file, which determines the styles of the Carbon framework, is located in the org.wso2.carbon.ui_<version-name>.jar/web/admin/css/ directory. You can edit this file as per your requirement. Alternatively, you can apply a completely new stylesheet to your framework instead of the default global.css stylesheet.

To apply a new style sheet to the carbon framework:

1. Copy your new CSS file to this same location.
2. Open the template.jsp file located in the org.wso2.carbon.ui_<version-name>.jar/web/admin/layout/ directory, which contains the main layout of the page and the default JavaScript libraries.
3. Replace global.css with the new style sheet by pointing the String globalCSS attribute to the new stylesheet file.

```
//Customization of UI theming per tenant
String tenantDomain = null;
String globalCSS = "../admin/css/global.css";
String mainCSS = "";
```

Changing the product specific styles and images

The styles and images unique to your product are located in the `org.wso2.<product-name>.styles_<version-number>.jar` folder. To modify product specific styles and images, take the following steps.

1. Copy the necessary images to the `org.wso2.<product-name>.styles_<version-number>.jar/web/styles/images/` directory. For example, if you want to change the product banner, add the new image file to this directory.
2. Open the `main.css` file located in the `org.wso2.<product-name>.styles_<version-number>.jar/web/styles/css/` directory.
3. To specify a new product banner, change the `background-image` attribute of `org.wso2.<product-name>.styles_<version-number>.jar/web/styles/css/main.css` file as follows:

```
/* ----- header styles ----- */
div#header-div {
    background-image: url( ../images/newproduct-header-bg.png );
    height:70px;
}
```

Note that the size of the images you use will affect the overall UI of your product. For example, if the height of the product logo image exceeds 28 pixels, you must adjust the `main.css` file in the `org.wso2.<product-name>.styles_<version-number>.jar/web/styles/css/` directory to ensure that the other UI elements of your product aligns with the product logo.

Starting the server

In the preceding steps, you have done the changes to the product interface after copying the JAR files to a separate location on your hard drive. Therefore, before you start your production server, these files must be correctly copied back to your production environment as explained below.

1. Compress the contents of the `org.wso2.carbon.ui_<version-number>.jar` and `org.wso2.<product-name>.styles_<product-version>.jar` folders into separate ZIP files.
2. Change the name of the ZIP file to `org.wso2.carbon.ui_<version-number>.jar` and `org.wso2.<product-name>.styles_<version-number>.jar` respectively.
3. Copy these two new JAR files to the `<PRODUCT_HOME>/repository/components/plugins/` directory in your product installation.
4. Start the server.

Using Analytics Spark Features in Other WSO2 Products

The server start-up scripts of [WSO2 Carbon-based](#) products include configurations required to start a Spark server instance. The following sections explain the additional changes that need to be done in order to use the required Analytics Spark features.

- [Install Spark features](#)
- [Editing start-up scripts](#)

Install Spark features

The Spark features to be installed are as follows. For detailed instructions for installing features, see [Installing and Managing Features](#).

Feature	Purpose

org.wso2.carbon.analytics.spark.server.feature	This feature contains the classes required to use Spark in a WSO2 Carbon environment.
org.wso2.carbon.spark.commons.feature	This feature contains Apache Spark related OSGI bundles.

Editing start-up scripts

Follow the procedure below to ensure that the Spark features you installed are initialized at product start-up.

For Windows

1. Download the `load-spark-env-vars.bat` file from [here](#) and add it to the `<PRODUCT_HOME>/bin` directory.
2. Add the following line to the start-up script of the product in the `<PRODUCT_HOME>\bin\wso2server.bat` file.

```
rem ----- loading spark specific variables
call %CARBON_HOME%\bin\load-spark-env-vars.bat
```

1. Download the `load-spark-env-vars.sh` file from [here](#) and add it to the `<PRODUCT_HOME>/bin` directory.
2. Add the following line to the start-up script of the product in the `<PRODUCT_HOME>\bin\wso2server.sh` file.

```
#load spark environment variables
. $CARBON_HOME/bin/load-spark-env-vars.sh
```

WSO2 DAS Performance Analysis

This section summarizes the results of performance tests carried out with the minimum fully distributed DAS deployment setup with RDBMS (MySQL) and HBase event stores separately.

DAS Performance Test Round 1: RDBMS

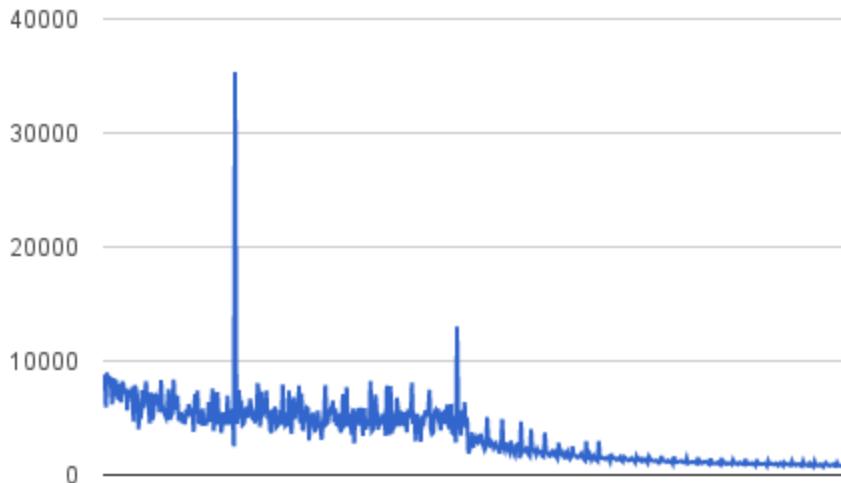
Infrastructure used

- c4.2xlarge Amazon EC2 instances as the DAS nodes
- One DAS node as the publisher
- A c3.2xlarge Amazon instance as the database node

Receiver node Data Persistence Performance

A reduction in the throughput is observed as shown below after 1200000 events in both DAS receiver nodes. This reduction is caused by limitations of MySQL. The receiver performance variation of the second node of the 2 node receiver cluster is as given below. As the initial buffer filling in receiver queues give very high receiver performance at the beginning of the event publishing, event rate after the first 1200000 events was considered for the following graph.

Node 2 throughput from 1200000 onwards



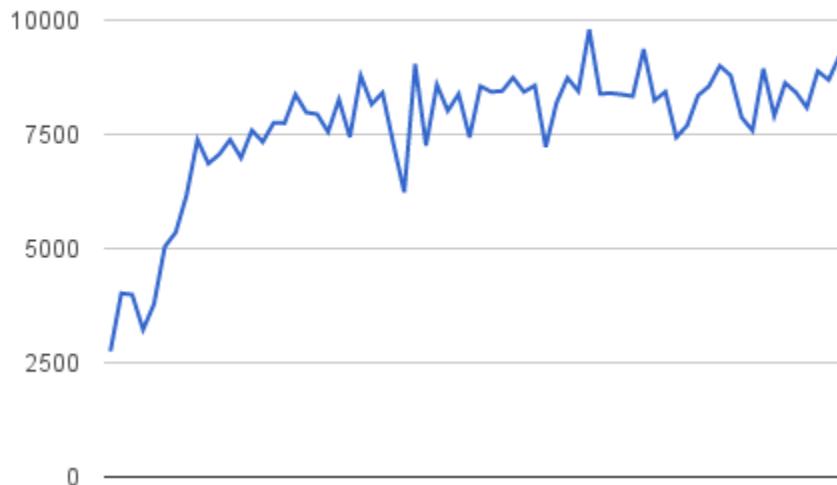
MySQL Breakpoint

After around 30 million events are published, a sudden drop can be observed in receiver performance that can be considered as the break point of MySQL event store. Another type of event store such as HBase event store should be used when the receiver performance has to be maintained unchanged.

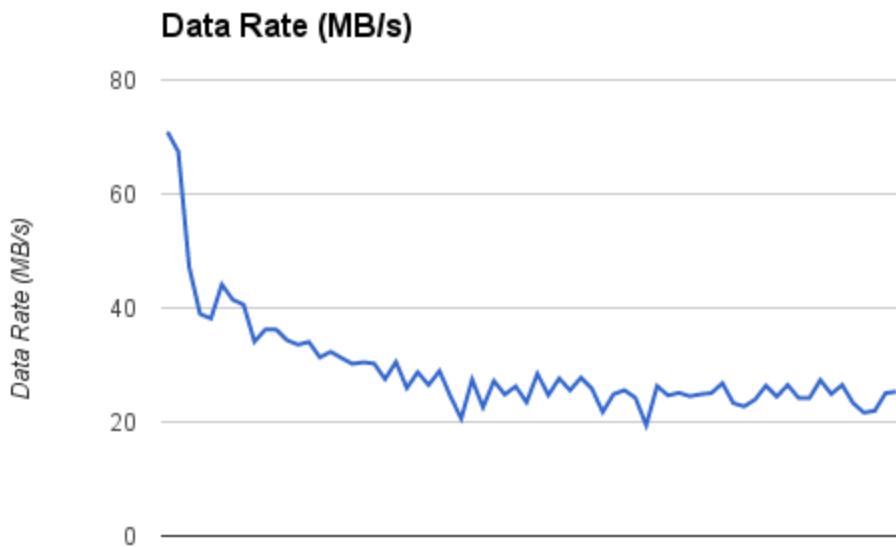
Testing with large events

The following results were obtained by testing the two-node HA (High Availability) DAS cluster with 10 million events published via the [Analyzing Wikipedia Data](#) sample. Each event in this sample contains several kilobytes that represent large events.

TPS



In the above graph, TPS represents the total number of events published per second. This stabilizes at about 8500 events per second.



The above graph shows the amount of data that is published per second (referred to as the data rate). The data rate published is significantly reduced at the initial stages due to the flow control mechanisms of the receiver. It stabilizes at around 25 MB per second.

With MySQL RDBMS event store

DAS data persistence was measured by publishing to 2 load balanced receiver nodes with MySQL database.

Sample	Number of Events	Mean Event Rate
Smart Home sample	100000000	5741 events per second
Wikipedia sample	15901127	4438 events per second

Analyzer Performance

The following topics describe the analyzer performance of WSO2 DAS.

With MySQL RDBMS event store

Spark analyzing performance (time to complete execution) was measured using a 2 node DAS analyzer cluster with MySQL database.

Time taken for each type of Spark query is given below.

Data set	Event Count	Query Type	Time Taken (seconds)
Smart Home	10000000	INSERT OVERWRITE TABLE cityUsage SELECT metro_area, avg(power_reading) AS avg_usage, min(power_reading) AS min_usage, max(power_reading) AS max_usage FROM smartHomeData GROUP BY metro_area	26 sec

Smart Home	10000000	INSERT OVERWRITE TABLE peakDeviceUsageRange SELECT house_id, (max(power_reading) - min(power_reading)) AS usage_range FROM smartHomeData WHERE is_peak = true AND metro_area = "Seattle" GROUP BY house_id	22 sec
Smart Home	10000000	INSERT OVERWRITE TABLE stateAvgUsage SELECT state, avg(power_reading) AS state_avg_usage FROM smartHomeData	21 sec
Smart Home	10000000	INSERT OVERWRITE TABLE stateUsageDifference SELECT a2.state, (a2.state_avg_usage-a1.overall_avg) AS avg_usage_difference FROM (select avg(state_avg_usage) as overall_avg from stateAvgUsage) as a1 join stateAvgUsage as a2	1 sec
Wikipedia	10000000	INSERT INTO TABLE wikiAvgArticleLength SELECT AVG(length) as avg_article_length FROM wiki	48 min
Wikipedia	10000000	INSERT INTO TABLE wikiContributorSummary SELECT contributor_username, COUNT(*) as page_count FROM wiki GROUP BY contributor_username	1 hour 45 min
Wikipedia	10000000	INSERT INTO TABLE wikiTotalArticleLength SELECT SUM(length) as total_article_chars FROM wiki	44 min
Wikipedia	10000000	INSERT INTO TABLE wikiTotalArticlePages SELECT COUNT(*) as total_pages FROM wiki	1 hour 17 min

DAS Performance Test Round 2: HBase Cluster

This test involved setting up a 10-node HBase cluster with HDFS as the underlying file system.

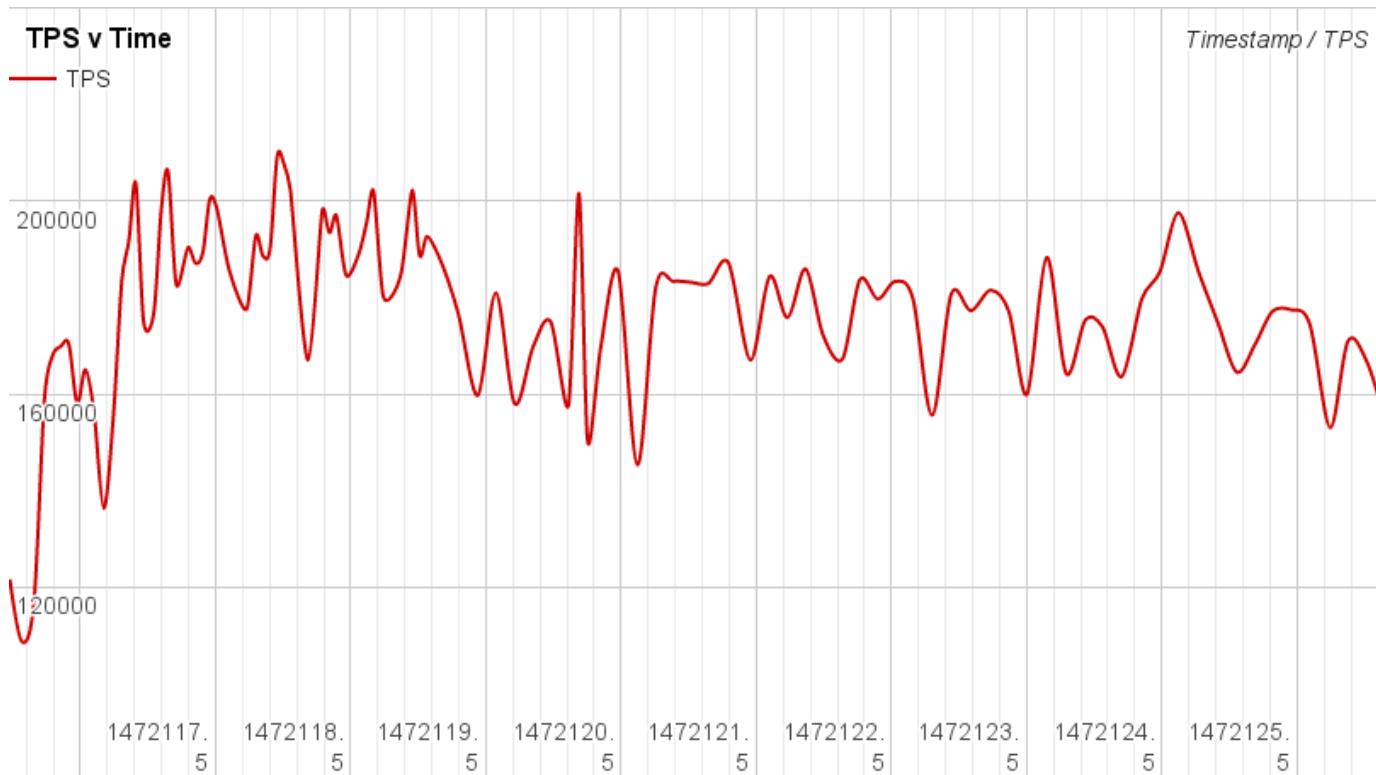
Infrastructure used

- 3 DAS nodes (variable roles: publisher, receiver, analyzer and indexer): c4.2xlarge
- 1 HBase master and Hadoop Namenode: c3.2xlarge
- 9 HBase Regionservers and Hadoop Datanodes: c3.2xlarge

Persisting 1 billion events from the Smart Home DAS Sample

This test was designed to test the data layer during sustained event publication. During testing, the TPS was around the 150K mark, and the memstore flush of the HBase cluster (which suspends all writes) and minor compaction operations brought it down in bursts. Overall, a mean of 96K TPS was achieved, but a steady rate of around 100-150K TPS as is achievable, as opposed to the current no-flow-control situation.

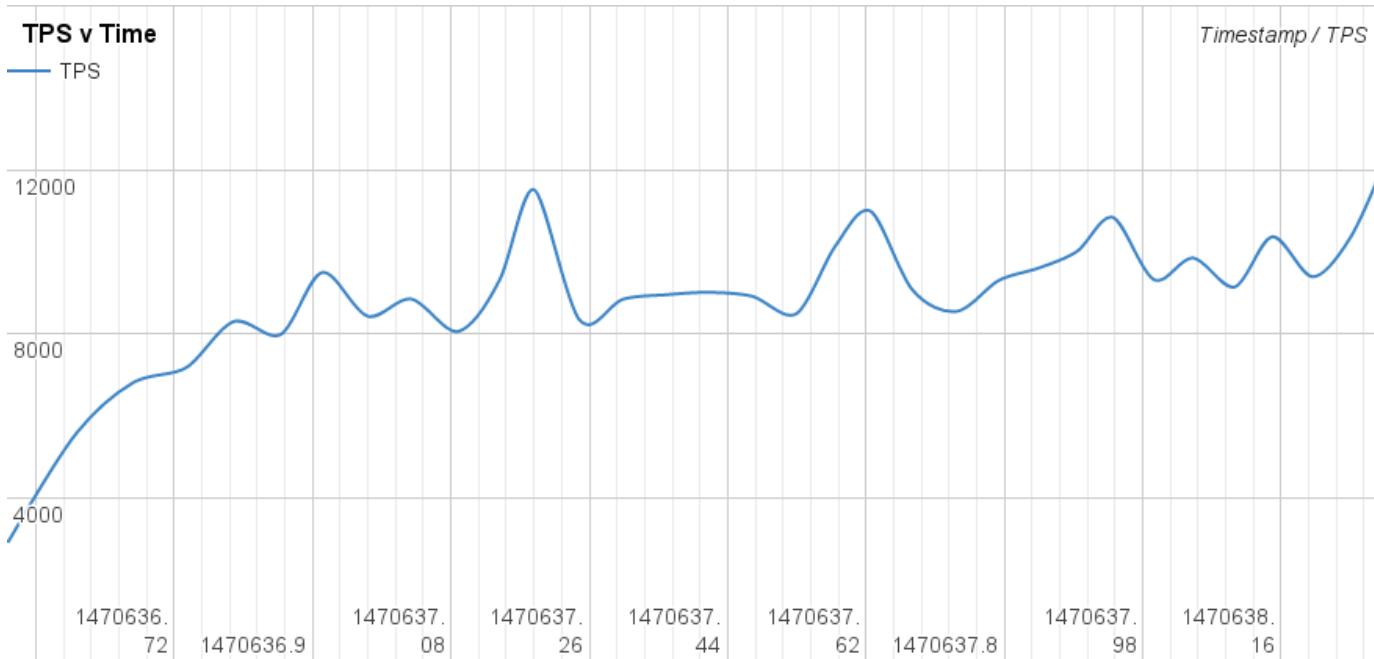
The published data took around 950GB on the Hadoop filesystem, taking the HDFS-level replication into account.



Events	10000000000
Time (in seconds)	10391.768
Mean TPS	96230.01591

Persisting the entire Wikipedia corpus

This test involved publishing the entirety of the Wikipedia dataset, where a single event comprises of one Wikipedia article (16.8M articles in total). Events vary greatly in size, with the mean being ~3.5KB. Here, a mean throughput of around 9K TPS was observed.



Events	16753779
Time (s)	1862.901
Mean TPS	8993.381291

Running Spark queries on the 1 billion published events

Spark queries from the Smart Home DAS sample were executed against the published data, and the analyzer node count was kept at 2 and 3 respectively for 2 separate tests. The SPARK JVMs were provided with following during the test.

- 1.36 processor cores
- 12GB of dedicated memory

The following results were observed.

- Over 1 million TPS on Spark for 2 analyzers
- About 1.3 million TPS for 3 analysers.

The DAS GET operations (on HBase) make use of the HBase data locality aspect. This has the potential to perform the GET operations fast compared to random access.

Query	2 Analyzers		3 Analyzers	
	Time(s)	Mean TPS	Time(s)	Mean TPS
INSERT OVERWRITE TABLE cityUsage SELECT metro_area, avg(power_reading) AS avg_usage,min(power_reading) AS min_usage,max(power_reading) AS max_usage FROM smartHomeData GROUP BY metro_area	958.80	1042968.20	741.15	1349250.90

INSERT OVERWRITE TABLE ct SELECT count(*) FROM smartHomeData	953.46	1048806.20	734.99	1360570.13
INSERT OVERWRITE TABLE peakDeviceUsageRange SELECT house_id, (max(power_reading) - min(power_reading)) AS usage_range FROM smartHomeData WHERE is_peak = true AND metro_area = "Seattle" GROUP BY house_id	975.06	1025581.77	751.27	1331073.47
INSERT OVERWRITE TABLE stateAvgUsage SELECT state, avg(power_reading) AS state_avg_usage FROM smartHomeData GROUP BY state	991.08	1009003.34	783.54	1276265.545

Running Spark queries on the Wikipedia corpus

Query	2 Analyzers		3 Analyzers	
	Time(s)	Mean TPS	Time(s)	Mean TPS
INSERT INTO TABLE wikiAvgArticleLength SELECT AVG(length) as avg_article_length FROM wiki	222.70	75234.03	167.27	100164.18
INSERT INTO TABLE wikiTotalArticleLength SELECT SUM(length) as total_article_chars FROM wiki	221.74	75554.76	166.92	100373.80
INSERT INTO TABLE wikiTotalArticlePages SELECT COUNT(*) as total_pages FROM wiki	221.80	75536.05	166.14	100842.18
INSERT INTO TABLE wikiContributorSummary SELECT contributor_username, COUNT(*) as page_count FROM wiki GROUP BY contributor_username	236.11	70958.52	181.42	92350.26

Single Node Local Clustered Setup Statistics

A fully distributed setup was tested locally with multiple JVMs, and with the following hardware infrastructure specifications.

Machine type: Laptop

RAM: 8GB

Processor: Intel(R) Core(TM) i7-3520M

Storage: Samsung SSD 850

The setup is a 2 node analyzer cluster with a MySQL database as the event store. Analyzer statistics (i.e., the time duration for each execution of the query) are given below.

Dataset	Event Count	Query Type	Time Taken

Wikipedia	15901127	INSERT INTO TABLE wikiContributorSummary SELECT contributor_username, COUNT(*) as page_count FROM wiki GROUP BY contributor_username	25 min
Wikipedia	15901127	INSERT INTO TABLE wikiTotalArticleLength SELECT SUM(length) as total_article_chars FROM wiki	25 min
Wikipedia	15901127	INSERT INTO TABLE wikiTotalArticlePages SELECT COUNT(*) as total_pages FROM wiki	25 min
Wikipedia	15901127	INSERT INTO TABLE wikiAvgArticleLength SELECT AVG(length) as avg_article_length FROM wiki	25 min

It was observed that the performance here is comparatively higher (taking into account that the setup consists of a single machine). This is mainly due to the DAS server and MySQL existing locally, and having no physical network I/O delays as a result. This allows the queries to be executed in an optimal manner.

Indexing Performance

In the following table, the `shardIndexRecordBatchSize` indicates the amount of index data (in bytes) to be processed at a time by a shard index worker.

Mode	Dataset	shardIndexRecordBatchSize	Replication Factor	Event Count	Time Taken (seconds)	Average TPS
Standalone	Wikipedia	10MB	NA	15901127	7975	1993.871724
Standalone	Wikipedia	20MB	NA	15901127	6765	2350.499187
Standalone	Smart Home	20MB	NA	20000000	1385	14440.43321
Minimum Fully Distributed	Wikipedia	20MB	1	15901127	6870	2314.574527
Minimum Fully Distributed	Wikipedia	20MB	0	15901127	7280	2184.220742

Working with Dashboards

The following topics provide reference information on customizing dashboards.

- [Adding a Custom Theme for a Dashboard](#)

Adding a Custom Theme for a Dashboard

Generally, you can use the default theme to create a dashboard. However, there may be situations where you may need a customized view of a particular dashboard, and need to maintain different custom themes per dashboard. In such situations, you can create your own custom theme and apply it to a particular dashboard.

Creating a custom theme for a specific dashboard

Follow the instructions below to create a custom theme for a specific dashboard:

1. Create a custom theme file.
Create a custom theme file as a Carbon archive file, which has the `.car` extension (e.g., `Custom-Theme.car`).

ar). Use dashboards/theme as the artifact type of the theme file. For example, the artifact.xml file for a custom theme is as follows:

```
<artifact name="custom-theme-sample" version="1.0.0" type="dashboards/theme"
serverRole="DataAnalyticsServer">
    <file>custom-theme-sample</file>
</artifact>
```

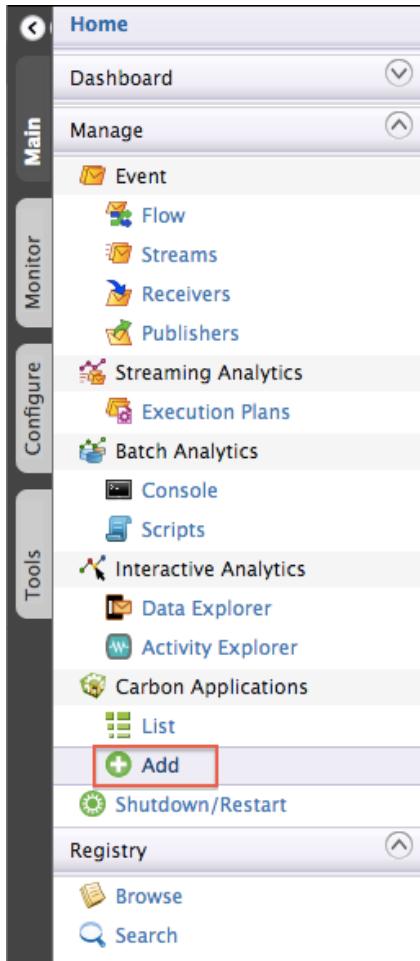
The folder structure for the custom theme file is as follows:

```
css
js   -----
    ues.js
    ues-prefs.js
    ues-gadgets.js
    ues-dashboards.js
    gadget-properties.js
    gadget-handlebar-helpers.js
    dashboard.js
    ds-ajax-api.js
templates ---
    includes -----
        portal-header.jag
        portal-footer.jag
        control-panel.jag
        light-dark-switcher.jag
        project-title.jag
```

The css folder can contain any stylesheets with the .css extension. However, the js and template folders should contain the files defined in the above structure.

2. Deploy the custom theme.

- a. Sign into the WSO2 Data Analytics Server (WSO2 DAS) Management Console.
<https://<hostname>:9443/carbon>
- b. Click **Main**, navigate to **Carbon Application** and click **Add**.



- c. Select the Carbon Application Artifact (.car) file and upload it.

Home > Manage > Carbon Applications > Add

Add Carbon Applications

UPLOAD CARBON APPLICATION

Carbon Application Artifact(.car) Custom_Theme.car

The uploaded custom theme appears as follows:

Carbon Applications List		
1 Running Carbon Applications.		
CARBON APPLICATIONS	VERSION	ACTIONS
Custom_Theme	1.0.0	

Applying the custom theme to a dashboard

You can use one of the following methods to apply your custom theme to a specific dashboard.

- When creating a dashboard, you can select the your custom theme.

Create a Dashboard

Name of your Dashboard *

URL *

Description

E.g. Monthly Sales Statistics

Theme

Next

- You can also change the existing theme of a dashboard via the settings view of that respective dashboard.

Dashboard Settings

Basics

URL /dashboards/test-custom-theme

Title *

Description

Description

Theme

Share Dashboard



Disabled

Personalize Dashboard



Disabled

Permissions

Viewers

Roles

Internal/test-custom-theme-viewer X

Editors

Roles

Internal/test-custom-theme-editor X

Owners

Roles

Internal/test-custom-theme-owner X

Save **Reset**

After creating a custom dashboard theme and applying the theme, you can preview the final view of the dashboard

using the dashboard view option.

FAQ

- I see an exception stating - java.io.IOException: Cannot run program "null/bin/java" when running DAS? What is going wrong?
- How can I scale up DAS?
- I only see one DAS distribution. How do I create a DAS receiver node/ analyzer node/ dashboard node?
- Can I send custom data/ events to DAS?
- How do I define a custom KPI in DAS?
- Can DAS do real time analytics
- I see that in the DAS samples, it writes the results to a RDBMS? Why does it do this?
- I get a read timeout in the analytics UI after executing a query?
- I am getting error "Thrift error occurred during processing of message..." Any idea why?
- I am getting an error "OpenAjax.hub.SecurityAlert.LoadTimeout" Any idea why?

I see an exception stating - java.io.IOException: Cannot run program "null/bin/java" when running DAS? What is going wrong?

This happens when you have not set the JAVA_HOME environment variable and pointed to the installed JRE location. This needs to be explicitly set in your environment.

How can I scale up DAS?

If you want to scale up DAS to receive a large amount of data, you can setup multiple receiver nodes fronted by a load balancer. If you want to scale up the dashboards (presentation layer), you can setup multiple dashboard nodes fronted by a load balancer.

I only see one DAS distribution. How do I create a DAS receiver node/ analyzer node/ dashboard node?

The DAS distribution will contain all the features you need. We prepare each node by uninstalling relevant features. You can do this by the feature installation/ uninstallation ability that comes with WSO2 DAS. If you want to create a receiver node, you uninstall the analytics feature and the dashboard feature, and you will have a receiver node.

Can I send custom data/ events to DAS?

Yes, you can. There is an SDK provided for this. For a detailed article on how to send custom data to DAS using this SDK, see [Creating Custom Data Publishers to BAM/CEP](#).

You can also send data to DAS using the REST API. For information on sending data to DAS using the REST API, see [REST APIs for Analytics Data Service](#).

How do I define a custom KPI in DAS?

The model for this is to first publish the custom data. After you send custom data to DAS, you need to define your analytics to match your KPI. To visualize the result of this KPI, you can write a gadget using HTML and JS or use the Gadget generation tool. For all artifacts related to defining a custom KPI, see [KPI definition and monitoring sample](#).

Can DAS do real time analytics

WSO2 DAS can process large data volumes in real time. This is done via WSO2 CEP components that are also installed in WSO2 DAS by default. The [WSO2 CEP server](#) is a powerful real time analytics engine capable of defining queries based on temporal windows, pattern matching and much more.

I see that in the DAS samples, it writes the results to a RDBMS? Why does it do this?

The DAS does this for 2 reasons. One is to promote a polyglot data architecture. It can be stored in a RDBMS or any other data store. The second is that there is extensive support for many 3rd party reporting tools such as Jasper, Pentaho, etc. already support RDBMSs. With this sort of support for a polyglot data architecture, any reporting engine or dashboard can be plugged into DAS without any extra customization effort.

I get a read timeout in the analytics UI after executing a query?

This happens when there is a large amount of data to analyze. The UI will timeout after 10 minutes, if the data to be processed takes more time than this.

I am getting error "Thrift error occurred during processing of message..." Any idea why?

If you are getting the following while trying to publish data from DAS mediator data agent then check whether you have specified the receiver and authentication ports properly and have not mixed them up. The default values are receiver port = 7611

authentication port = 7711

```
TID: [0] [BAM] [2012-11-28 22:46:40,102] ERROR {org.apache.thrift.server.TThreadPoolServer} - Thrift error occurred during processing of message. {org.apache.thrift.server.TThreadPoolServer}
org.apache.thrift.protocol.TProtocolException: Bad version in readMessageBegin
at org.apache.thrift.protocol.TBinaryProtocol.readMessageBegin(TBinaryProtocol.java:208)
at org.apache.thrift.TBaseProcessor.process(TBaseProcessor.java:22)
at org.apache.thrift.server.TThreadPoolServer$WorkerProcess.run(TThreadPoolServer.java:176)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
at java.lang.Thread.run(Thread.java:619)
```

I am getting an error "OpenAjax.hub.SecurityAlert.LoadTimeout" Any idea why?

The OpenAjax.hub.SecurityAlert.LoadTimeout error can occur when you have pub-sub gadgets in your dashboard. Shindig uses the pubsub-2 feature of OpenAjax.hub. Timeout is used in OpenAjax.hub to avoid frame phishing attacks. If an iframe with the gadget is not loaded within a particular time period, it throws this security exception and immediately stops the iframe loading process to avoid iframe phishing.

The load timeout is 15 secs for all the pub-sub gadgets regardless of complexity of the gadgets. As Shindig does not provide a method to configure timeouts, WSO2 has extended the shindig features to provide this facility.

WSO2 has changed the default timeout to 60 secs, because 15 secs is not sufficient in a production environment. Furthermore, WSO2 allows you to decide and configure the timeout per gadget using either one of the following methods.

- Include a `timeoutInterval` (in milliseconds) under `settings` in the `gadget.json` file.

```
"settings": {
    "timeoutInterval":100000
}
```

- In the gadget configuration panel, you can set the timeout interval by manually entering the timeout Interval in milliseconds.

Glossary

[Component](#) | [Data Access Layer](#) | [Latency](#) | [Port offset](#) | [SOAP](#) | [Throughput](#) | [WSO2 DAS Management Console](#)

Component

Components in the Carbon platform add functionality to all WSO2 Carbon-based products. For example, the statistics component enables users to monitor system and service level statistics. A component in the Carbon platform is made up of one or more [OSGi](#) bundles, which is the modularization unit in OSGi similar to a JAR file in Java. For example, the statistics component contains two bundles: one is the back-end bundle that collects, summarizes, and stores statistics, and the other is the front-end bundle, which presents the data to the user through a user-friendly interface. This component-based architecture of the WSO2 Carbon platform gives developers flexibility to build efficient and lean products that best suit their unique business needs simply by adding and removing components.

Data Access Layer

The data access layer persists the data processed by WSO2 DAS and consists of two components named Analytics Record Store and Analytics File System.

The Analytics Record Store handles the storage of records that are received by WSO2 DAS in the form of events. The Analytics File System handles the storage of index data.

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](#) 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.

Port offset

The port offset feature allows you to run multiple WSO2 products, multiple instances of a WSO2 product, or multiple WSO2 product clusters on the same server or virtual machine (VM). The port offset defines the number by which all ports defined in the runtime such as the HTTP/S ports will be offset. For example, if the HTTP port is defined as 9763 and the portOffset is 1, the effective HTTP port will be 9764. Therefore, for each additional WSO2 product, instance, or cluster you add to a server, set the port offset to a unique value (the default is 0).

Port offset can be passed to the server during startup as follows:
`./wso2server.sh -DportOffset=3`

Alternatively, you can set it in the Ports section of `<PRODUCT_HOME>/repository/conf/carbon.xml` as follows:
`<Offset>3</Offset>`

Important

The tool boxes that come with samples use an embedded H2 database to persist summarized data. They only work with the default DAS installation. If you change the default settings (e.g., port offset values, H2 database settings), you should also change the corresponding Hive scripts.

For example, the Hive script associated with the default KPI Monitoring sample has Cassandra port set as 9160. If you change the port offset, you should manually change the Cassandra port too, by following these steps.

steps:

1. After installing the tool box, log in to the management console, select **List** sub menu under **Analytics** menu. This will open the **Available Scripts** window.
2. Set "cassandra.port" = "9161" in the phone_retail_store_script and Save.

SOAP

An XML-based, extensible message envelope format, with "bindings" to underlying protocols. The primary protocols are HTTP and HTTPS, although bindings for others, including SMTP and XMPP, have been written.

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 DAS server. The throughput is affected by the performance configurations (see [Performance Tuning](#) 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.

WSO2 DAS Management Console

WSO2 DAS Management Console is a Web based control panel powered by JSP and AJAX which enables system administrators to interact with a running the DAS instance without touching any underlying configuration files. The Management Console allows the users to command and control proxy services, sequences, transports, local entries, registry, modules, endpoints and much more.

Getting Support

In addition to this documentation, there are several ways to get help as you work on WSO2 products.

	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.
	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/ .
	Ask questions in the user forums at http://stackoverflow.com . Ensure that you tag your question with appropriate keywords such as <i>WSO2</i> 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 .
	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).

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.

Tutorials

This section covers the following use cases of WSO2 DAS.

- [Create Event Stream](#)
- [Persist Event Stream](#)
- [Simulate Events for a Persisted Event Stream](#)
- [Search for Records via the Management Console](#)

Create Event Stream

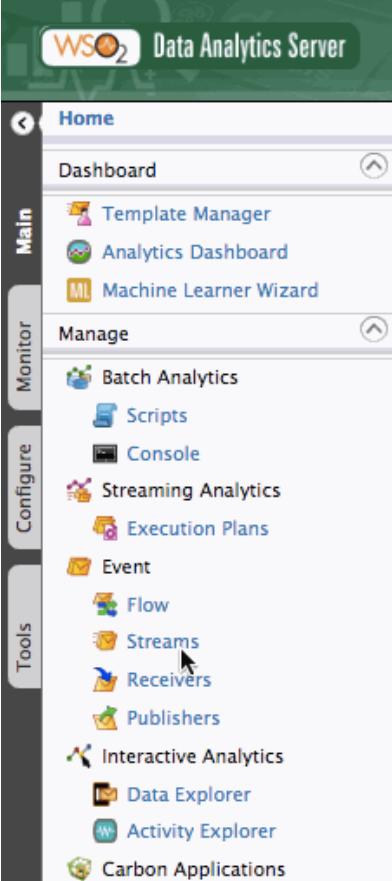
WSO2 DAS processes data as events, and also interact with external systems using events. An event stream is a sequence of events categorized into specific types.

This tutorial demonstrates how to create an event stream in WSO2 DAS to persist information relating to books in a book store.

For a description of **concepts** you need to understand before creating an event stream, see [Understanding Event Streams and Event Tables](#).

Follow the steps below to create an event stream in WSO2 DAS.

1. Access the WSO2 DAS Management Console using the `https://<DAS_HOST>:<DAS_PORT>/carbon/` URL, and then log in with your credentials.
2. In the **Main** tab, click **Streams** to open the **Available Event Streams** page, and click **Add Event Stream**. This opens the **Define New Event Stream** page.



The screenshot shows the WSO2 Data Analytics Server Management Console interface. The top navigation bar includes the WSO2 logo, the title "Data Analytics Server", and a "Signed-in as: admin@carbon.super" message. The left sidebar has tabs for "Main", "Monitor", "Configure", and "Tools". Under the "Main" tab, there are sections for "Dashboard", "Template Manager", "Analytics Dashboard", and "Machine Learner Wizard". Under "Manage", there are sections for "Batch Analytics" (with "Scripts", "Console", "Streaming Analytics", "Execution Plans", "Event", "Flow", and "Streams" listed), "Interactive Analytics" (with "Data Explorer" and "Activity Explorer"), and "Carbon Applications". A mouse cursor is hovering over the "Streams" link under the "Batch Analytics" section. The main content area displays the "WSO2 Data Analytics Server Home" page with a welcome message and three tables: "Server", "Operating System", and "Operating System User".

Server	
Host	10.100.5.73
Server URL	local:/services/
Server Start Time	2016-10-31 11:16:07
System Up Time	0 day(s) 0 hr(s) 5 min(s) 13 sec(s)
Version	3.1.0
Repository Location	file:/Users/rukshaniweerasinha/Downloads/ProductPacks/Ar3.1.0/repository/deployment/server/

Operating System	
OS Name	Mac OS X
OS Version	10.9.5

Operating System User	
Country	US
Home	/Users/rukshaniweerasinha

3. Create a new event stream by entering information as shown below.

[Home](#)[Help](#)

Define New Event Stream

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	BookStore	<small>⑦ Name of the Event Stream</small>
Event Stream Version*	1.0.0	<small>⑦ Version of the event stream (Eg : 1.0.0)</small>
Event Stream Description	<small>⑦ Description of the event stream</small>	
Event Stream Nick-Name	<small>⑦ Nick name of the event stream</small>	

Stream Attributes

Meta Data Attributes

Attribute Name	Attribute Type	Actions
Transaction_ID	string	Delete

Attribute Name : Attribute Type :

Correlation Data Attributes

Attribute Name	Attribute Type	Actions
Title	string	Delete
Author	string	Delete
Price	float	Delete
Quantity	int	Delete
Date	string	Delete
Discount	float	Delete

Attribute Name : Attribute Type :

Payload Data Attributes

Attribute Name	Attribute Type	Actions
Value	float	Delete
_timestamp	long	Delete

Attribute Name : Attribute Type :

Parameter	Value
Event Stream Name	BookStore
Event Stream Version	1.0.0

Attributes

Attribute	Attribute Type
Transaction_ID	string

Title	string
Author	string
Price	float
Quantity	int
Date	string
Discount	float
Value	float
_timestamp	long

The `_timestamp` attribute needs to be defined in this tutorial as a payload attribute of the `long` data type in order to override the timestamp automatically assigned to each event by the system. This allows you to assign any timestamp of your choice.

- Click **Add Event Stream**. The new event stream is added to the list of all available event streams as shown below.

[Home](#) > [Manage](#) > [Event](#) > [Streams](#)

[Help](#)

Available Event Streams

[!\[\]\(3fef6148fcd98d3ed4b9784300d8ed75_img.jpg\) Add Event Stream](#)

1 Event streams available

Event Stream Id	Event Stream Description	Actions
BookStore:1.0.0		 Simulate  Delete  Edit

Persist Event Stream

WSO2 DAS has a Data Access Layer (DAL) in which events handled are persisted for future use. This allows you to carry out batch analytics and interactive analytics for the events published in DAS.

In this tutorial, you persist an event stream in order to perform batch analytics and interactive analytics for events directed to it.

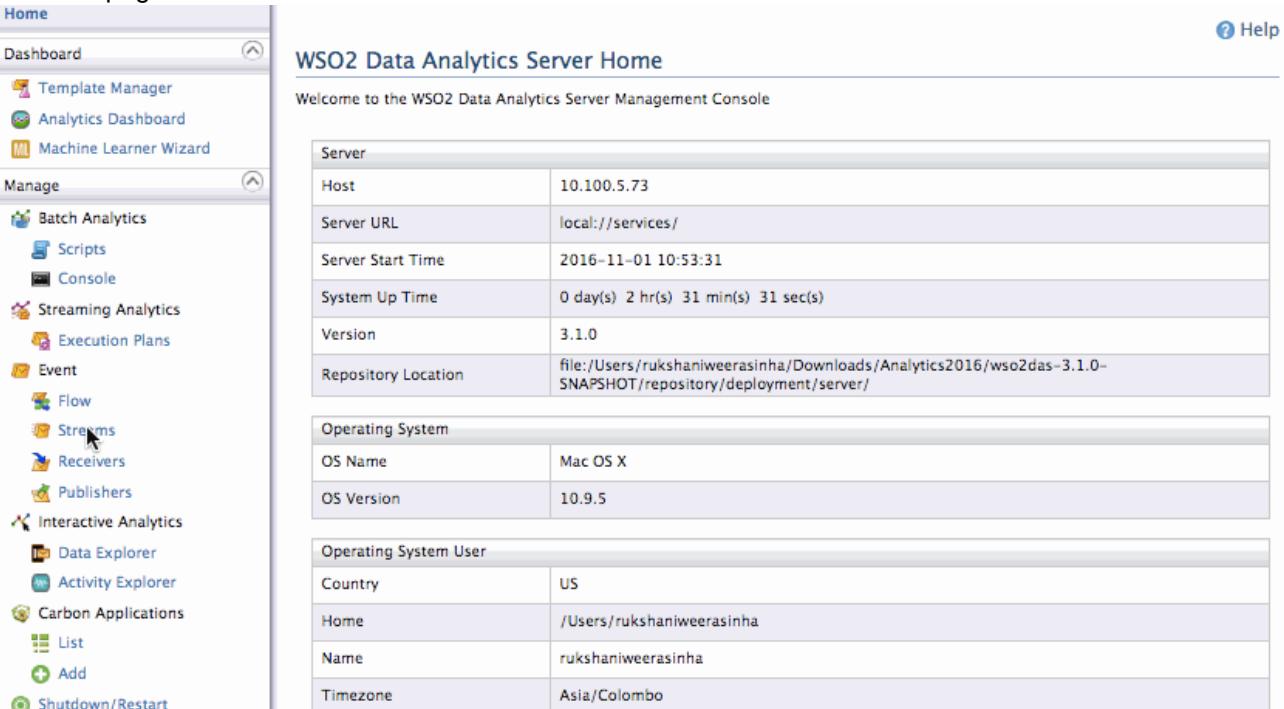
Click the following topics for a description of the **concepts** that you need to understand when persisting an event stream.

- [Configuring Data Persistence](#)
- [Implementation With Different Database Types](#)
- [Accessing Persisted Data](#)
- [Persisting Data for Batch Analytics](#)
- [Persisting Data for Interactive Analytics](#)

Before trying this tutorial, create the BookStore event stream following the instructions in the [Create Event Stream tutorial](#).

Follow the steps below to persist the BookStore event stream.

1. In the WSO2 DAS Management Console, go to the **Main** tab and click **Streams** to open the **Available Event Streams** page.



The screenshot shows the WSO2 Data Analytics Server Home page. On the left, there is a sidebar with various management options like Dashboard, Template Manager, Analytics Dashboard, Machine Learner Wizard, and several event-related sections (Batch Analytics, Streaming Analytics, Event, Interactive Analytics). The 'Streams' option under the Event section is highlighted with a cursor. The main content area displays the 'Available Event Streams' table with the following data:

Available Event Streams	
Stream Name	Description
BookStore	Bookstore event stream

Below the table, there are three sections: 'Server', 'Operating System', and 'Operating System User', each containing a table with specific details.

Server	
Host	10.100.5.73
Server URL	local://services/
Server Start Time	2016-11-01 10:53:31
System Up Time	0 day(s) 2 hr(s) 31 min(s) 31 sec(s)
Version	3.1.0
Repository Location	file:/Users/rukshaniweerasinha/Downloads/Analytics2016/wso2das-3.1.0-SNAPSHOT/repository/deployment/server/

Operating System	
OS Name	Mac OS X
OS Version	10.9.5

Operating System User	
Country	US
Home	/Users/rukshaniweerasinha
Name	rukshaniweerasinha
Timezone	Asia/Colombo

2. Click **Edit** for the XXX event stream to view it in the **Edit Event Stream** page, and then click **Next [Persist Event]**.

Home

[Help](#)

Edit Event Stream

Enter Event Stream Details [switch to source view](#)

Event Stream Name*	<input type="text" value="BookStore"/> <small>⑦ Name of the Event Stream</small>
Event Stream Version*	<input type="text" value="1.0.0"/> <small>⑦ Version of the event stream (Eg : 1.0.0)</small>
Event Stream Description	<input type="text"/>
Event Stream Nick-Name	<input type="text"/> <small>⑦ Nick name of the event stream</small>

Stream Attributes

Meta Data Attributes

Attribute Name	Attribute Type	Actions
Transaction_ID	string	Delete

Attribute Name : Attribute Type :

Correlation Data Attributes

Attribute Name	Attribute Type	Actions
Title	string	Delete
Author	string	Delete
Price	float	Delete
Quantity	int	Delete
Date	string	Delete
Discount	float	Delete

Attribute Name : Attribute Type :

Payload Data Attributes

Attribute Name	Attribute Type	Actions
Value	float	Delete

3. Select check boxes for attributes as follows.

None of these check boxes cannot be selected for the _timestamp attribute.

[Home](#)[Help](#)

Edit Event Stream

Enter Event Stream Details

<input checked="" type="checkbox"/> Persist Event Stream
Record Store EVENT_STORE

Meta Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Transaction_ID	STRING	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Correlation Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Title	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Author	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Price	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Quantity	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	Date	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Discount	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Payload Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	Is a Facet
<input checked="" type="checkbox"/>	Value	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	_timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes

No arbitrary data attributes are defined

Attribute Name :	Attribute Type :	Primary Key :	Index Column :	Score Param :	Is a Facet :	<input type="button" value="Add"/>
<input type="text"/>	<input type="button" value="INTEGER"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="button"/>

Advanced Help

[Back](#) | [Save Event Stream](#)

- **Persist Event Stream:** Select this check box in order to persist all the data handled by the BookStore event stream.
- **Persist Attribute:** Select this check box in the column header of each attribute type to ensure that the values for all the attributes in the event stream are persisted.
- **Primary Key:** Select this check box for the Transaction_ID attribute. As a result, the transaction ID serves as the unique ID for each row representing an event in the BookStore event table.
- **Index Column:** Select this check box for all the attributes to make it possible to search by any attribute.
- **Score Param:** Select this check box for the following attributes.
 - Price
 - Discount
- **Is a Facet:** Select this check box for the following attributes. This allows you to carry out advanced searches using these attributes by extracting their sub categories.
 - Title
 - Author
 - Date

4. Click **Save Event Stream** to save the changes.

Simulate Events for a Persisted Event Stream

Event simulation involves creating events in a predefined event stream by assigning values to the attributes in that event stream.

This tutorial demonstrates how to send a series of single events to a persisted event stream using the Event Stream Simulator tool.

Click the following topics for a description of the **concepts** that you need to understand when simulating single events.

- [Publishing Data Using Event Simulation](#)
- [Sending a Single Event by Entering Data](#)

Before you try this tutorial, do the following.

1. Create the BookStore event stream following the instructions in the [Create Event Stream tutorial](#).
2. Persist the BookStore event stream following the instructions in the [Persist Event Stream tutorial](#).

Follow the steps below to publish 15 single events in a persisted event stream.

1. In the WSO2 DAS Management Console, click on the **Tools** tab. Then click **Event Simulator** to open the **Event Stream Simulator** page.

The screenshot shows the WSO2 Data Analytics Server Management Console. The top navigation bar includes the WSO2 logo, 'Data Analytics Server', and tabs for 'Management Console' (Signed-in as: admin@carbon.super | Sign-out | Docs | About). The left sidebar has tabs for Main, Monitor, Configure, Test, and Tools. Under 'Tools', the 'Event Stream Simulator' option is highlighted. The main content area displays the 'WSO2 Data Analytics Server Home' page with sections for 'Server' (Host: 10.100.5.73, Server URL: local://services/, Server Start Time: 2016-11-03 09:42:17, System Up Time: 0 day(s) 2 hr(s) 22 min(s) 51 sec(s), Version: 3.1.0, Repository Location: file:/Users/rukshaniweerasinha/Downloads/ProductPacks/AnalyticsSetup/wso2das-3.1.0/repository/deployment/server/), 'Operating System' (OS Name: Mac OS X, OS Version: 10.9.5), 'Operating System User' (Country: US, Home: /Users/rukshaniweerasinha, Name: rukshaniweerasinha, Timezone: Asia/Colombo), and 'Java VM' (Java Home: /Library/Java/JavaVirtualMachines/jdk1.8.0_65.jdk/Contents/Home/jre, Java Runtime Name: Java(TM) SE Runtime Environment, Java Version: 1.8.0_65, Java Vendor: Oracle Corporation, Java VM Version: 25.65-b01).

2. In the **Event Stream Name** field, select the XXX event stream. The attributes in the stream are displayed as demonstrated below.

[Home > List](#)

[Help](#)

Event Stream Simulator

Send single event

Event Stream Name *	<input type="button" value="select event stream"/>
Stream Attributes	
<input type="button" value="Send"/> <input type="button" value="Clear"/>	

Send multiple events

Input Data by File			
File	Stream Configuration	Delay between events(ms)	Action
No file has been uploaded			
<input type="button" value="Choose File"/> No file chosen		<input type="button" value="upload"/>	

3. Simulate 10 events as given in the table below. Click **Send** to send each event.

Transaction_ID	Title	Author	Price	Quantity	Date	Discount
BSP0000001	T h e Invisible Man: A Grotesque Romance	Arthur,C,Clarke	6.66	12	2016-11-03	0
BSP0000002	Childhood's End	Arthur,C,Clarke	5.50	12	2016-10-06	0
BSP0000003	Oliver Twist	Charles,Dickens	6.96	20	2016-09-01	0
BSP0000004	Emma	Jane,Austen	6.73	20	2017-01-05	0
BSP0000005	Memoirs of a Geisha	Arthur,Golden	9.28	10	2017-02-02	0
BSP0000006	The Origin of Species	Charles,Darwin	6.25	20	2016-10-31	0

BSP0000007	The Adventures of Sherlock Holmes	Arthur,Conan,Doyle	6.00	15	2017-01-30	0.25
BSP0000008	A Study in Scarlet	Arthur,Conan,Doyle	5.00	15	2016-11-03	0.25
BSP0000009	Sherlock Holmes: The Complete Novels and Stories	Arthur,Conan,Doyle	9.50	30	2016-10-06	0.25
BSP0000010	Gone With the Wind	Margaret,Mitchell	6.83	20	2016-09-01	0
BSP0000011	Mary Called Magdalene	Margaret,George	7.00	10	2016-10-02	0.50
BSP0000012	Memoirs of Cleopatra	Margaret,George	7.00	10	2016-08-29	0.50
BSP0000013	Elizabeth I	Margaret,George	7.00	10	2017-01-30	0.50
BSP0000014	Sinatra: The Life	Anthony,Summers	10.00	5	2016-10-31	0
BSP0000015	1984	George,Orwell	6.99	20	2017-01-02	0

The comma (,) separated values entered for the **Author** attribute define a hierarchy of categories when searching by the Author facet.

e.g., In the above table, the following three authors have the same first name.

- Arthur,Conan,Doyle
- Arthur,C,Clarke
- Arthur,Golden

Here, Arthur is considered a main category of the Author facet. Conan, C, and Doyle are subcategories of the Arthur category. Doyle is a subcategory of the Conan subcategory.

This is further demonstrated in the [Search for Records Via the Management Console](#) tutorial.

Search for Records via the Management Console

The search functionality of WSO2 DAS is powered by Apache Lucene. It allows you to search for events persisted in event streams in different ways based on the way in which the event streams are persisted.

This tutorial demonstrates the different methods of searching for events using the WSO2 DAS Management Console.

Click the following topics for a description of the **concepts** that you need to know when searching for records.

- [Searching for Data](#)
- [Configuring Indexes](#)
- [Data Explorer](#)

Before you try this tutorial, do the following.

1. Create the BookStore event stream following the instructions in the [Create Event Stream tutorial](#).
2. Persist the BookStore event stream following the instructions in the [Persist Event Stream tutorial](#).
3. Simulate appropriately indexed events for the BookStore event stream following the instructions in the [Simulate Events for a Persisted Event Stream tutorial](#).

This tutorial demonstrates how to search for events received by a selected event stream in different method supported in WSO2 DAS.

1. In the WSO2 DAS Management Console, go to the **Main** tab and click **Data Explorer** to open the **Data Explorer** page.

The screenshot shows the WSO2 Data Analytics Server Management Console. The left sidebar has a 'Main' tab selected, showing navigation links for Home, Dashboard, Template Manager, Analytics Dashboard, Machine Learner Wizard, Batch Analytics, Scripts, Console, Streaming Analytics (Execution Plans, Event, Flow, Streams, Receivers, Publishers), Interactive Analytics (Data Explorer, Activity Explorer), Carbon Applications (List, Add), and Shutdown/Restart. Below this is a Registry section with Browse and Search links. The right panel title is 'Management Console' with a 'Signed-in as: admin@carbon.super | Sign-out | Docs | About' link. The main content area shows the 'Data Explorer' page with a breadcrumb 'Home > Manage > Interactive Analytics > Data Explorer'. It features a 'Search' section with a 'Table Name*' field containing 'Select a Table', a 'Search' button, and a 'Reset' button.

2. In the Table Name field, select **BOOKSTORE.**

The screenshot shows the 'Data Explorer' page with the 'Table Name*' field set to 'Select a Table'. A dropdown menu is open, listing three options: 'TEST_STREAM', 'TESTING_STREAM', and 'BOOKSTORE'. The 'BOOKSTORE' option is visible at the bottom of the list.

3. Click **Search.**

Search

Table Name*	BOOKSTORE	Schedule Data Purging
Maximum Result Count	1000	
Search	<input type="radio"/> By Date Range <input type="radio"/> By Primary Key <input type="radio"/> By Query	
<input type="button" value="Search"/> <input type="button" value="Reset"/>		

Results

Note: Total record count for the table is not available.

ved.

As demonstrated above, this populates the **Results** table with the 15 records you created in the [Simulate Events tutorial](#).

4. To search for transactions that occurred during the time period 01-10-2016 to 31-12-2016, select the **By Range** option. Select 01-10-2016 for the **From** field, and 31-12-2016 for the **To** field. Then click **Search**.

Search

Table Name*	BOOKSTORE	Schedule Data Purging
Maximum Result Count	1000	
Search	<input checked="" type="radio"/> By Date Range <input type="radio"/> By Primary Key <input type="radio"/> By Query	
From: <input type="text"/> To: <input type="text"/>		
<input type="button" value="Search"/> <input type="button" value="Reset"/>		

Results

Note: Total record count for the table is not available.

The following 6 records are displayed in the **Results** table.

Results										
Note: Total record count for the table is not available.										
BOOKSTORE										
	meta_Transaction_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp	
1	BSP0000002	Childhood's End	Arthur,C,Clarke	5.5	12	2016-10-06	0.0	66.0	2016-10-06 13:37:05 IST	
2	BSP0000009	Sherlock Holmes: The Complete Novels and Stories	Arthur,Conan,Doyle	9.5	30	2016-10-06	0.25	285.0	2016-10-06 13:37:06 IST	
3	BSP0000006	The Origin of Species	Charles,Darwin	6.25	20	2016-10-31	0.0	125.0	2016-10-31 13:37:05 IST	
4	BSP0000014	Sinatra: The Life	Anthony,Summers	10.0	5	2016-10-31	0.0	50.0	2016-10-31 13:37:07 IST	
5	BSP0000001	The Invisible Man: A Grotesque Romance	Arthur,C,Clarke	6.66	12	2016-11-03	0.0	79.92	2016-11-03 13:37:05 IST	
6	BSP0000008	A Study in Scarlet	Arthur,Conan,Doyle	5.0	15	2016-11-03	0.25	75.0	2016-11-03 13:37:06 IST	

<< < 1 2 ... 99 100 > >> Go to page: 1 Row count: 10 Showing 1-10 of 1000

5. To search for the transaction with ID BSP0000004, select the **By Primary Key** option. Enter BSP0000004 in the **meta_Transaction_ID** field and click **Search**.

Home > Manage > Interactive Analytics > Data Explorer Help

Data Explorer

Search	
Table Name*	BOOKSTORE <input type="button" value="Schedule Data Purging"/>
Maximum Result Count	1000 <input type="button"/>
Search	<input type="radio"/> By Date Range <input checked="" type="radio"/> By Primary Key <input type="radio"/> By Query <input type="text" value="meta_Transaction_ID"/>
<input type="button" value="Search"/> <input type="button" value="Reset"/>	

The following record is displayed in the **Results** table.

Results										
Total Records: 0										
BOOKSTORE										
	meta_Transaction_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp	
1	BSP0000004	Emma	Jane,Austen	6.73	20	2017-01-05	0.0	134.6	2017-01-05 13:37:05 IST	

6. To view all available authors of whom the first name is Arthur, do the following.

- a. Select the **By Query** option. As a result, the **Select a Facet** list is displayed.

Home > Manage > Interactive Analytics > Data Explorer Help

Data Explorer

Search	
Table Name*	BOOKSTORE <input type="button" value="Schedule Data Purging"/>
Maximum Result Count	1000 <input type="button"/>
Search	<input type="radio"/> By Date Range <input checked="" type="radio"/> By Primary Key <input type="radio"/> By Query <input type="text" value="Search Query"/> <input type="button" value="Select a Facet"/>

- b. In the **Select a Facet** list, select **correlation_Author**. As a result, the **correlation_Author** field.

The screenshot shows the Data Explorer search interface. In the 'Search Query' field, there is a dropdown menu labeled 'Select a Facet' containing two options: 'correlation_Author' and 'correlation_Date'. The 'correlation_Author' option is highlighted.

- c. In the first list of the **correlation_Author** field, select **Arthur**. As a result, the second list for the **correlation_Author** field is displayed.

The screenshot shows the Data Explorer search interface. The 'correlation_Author' dropdown menu is expanded, displaying five names: Arthur, Margaret, Charles, Jane, and Conan. The 'Select a category' option is also visible at the top of the list.

- d. Expand the second list. The second names/initials Conan, C and Golden are displayed as demonstrated below.

The screenshot shows the Data Explorer search interface. The 'correlation_Author' dropdown menu is expanded, showing 'Arthur' selected. Below it, the 'Select a category' option is highlighted, and the list includes Conan, C, and Golden.

7. To search for books by the author Arthur Conan Doyle, select the **By Query** option and select the **correlation_Author** facet. In the **correlation_Author** field, select **Arthur**, and select other names from the sub category lists as shown below. Then click **Search**.

Data Explorer

Search

Table Name* **BOOKSTORE** Schedule Data Purging

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

Search Query
correlation_Author

correlation_Author Arthur Conan Doyle Select a category Remove

Search **Reset**

The following records are displayed in the **Results** table.

- To search for transactions where the price was 7.00, select the **By Query** option. Enter correlation_Price : 7.00 in the **Search** field. Then click the **Search** button.

Data Explorer

Search

Table Name* **BOOKSTORE** Schedule Data Purging

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

Search **Reset**

The following records are displayed in the **Results** table.

Results

Total Records: 3 (94 ms)

BOOKSTORE

	meta_Transaction_ID	correlation_Title	correlation_Author	correlation_Price	correlation_Quantity	correlation_Date	correlation_Discount	Value	_timestamp
1	BSP0000011	Mary Called Magdalene	Margaret,George	7.0	10	2016-10-02	0.5	70.0	2017-10-02 13:37:05 IST
2	BSP0000013	Elizabeth I	Margaret,George	7.0	10	2017-01-30	0.5	70.0	2017-01-30 13:37:05 IST
3	BSP0000012	Memoirs of Cleopatra	Margaret,George	7.0	10	2016-08-29	0.5	70.0	2017-08-29 13:37:05 IST

<< < 1 > >> Go to page: 1 Row count: 10 Showing 1-3 of 3