



Data Analytics Server

Documentation



WSO2 Data Analytics Server

Documentation

Version 3.0.0

Table of Contents

1. Upgrading from the Previous Release	6
2. WSO2 Data Analytics Server Documentation	6
2.1 About DAS	7
2.1.1 Introducing DAS	8
2.1.2 DAS Features	10
2.1.3 Architecture	11
2.1.4 About this Release	13
2.2 Getting Started	14
2.2.1 Quick Start Guide	15
2.2.2 Downloading the Product	38
2.2.3 Installation Prerequisites	38
2.2.4 Installing the Product	40
2.2.4.1 Installing on Linux	41
2.2.4.2 Installing on Windows	42
2.2.4.3 Installing as a Windows Service	44
2.2.4.4 Installing as a Linux Service	50
2.2.5 Building from Source	52
2.2.6 Upgrading from WSO2 BAM 2.5.0	53
2.2.7 Running the Product	57
2.2.8 Get Involved	60
2.2.8.1 WSO2 GitHub Repositories	63
2.2.9 WSO2 DAS Production Setup Checklist	66
2.3 User Guide	66
2.3.1 Understanding Event Streams and Event Tables	67
2.3.1.1 Configuring Data Persistence	75
2.3.1.1.1 Implementation With Different Database Types	80
2.3.1.1.2 Accessing Persisted Data	102
2.3.2 Collecting Data	104
2.3.2.1 Publishing Data to DAS	104
2.3.2.1.1 Publishing Data Using Event Simulation	104
2.3.2.1.2 Publishing Data Using Java Client Through REST API	107
2.3.2.1.3 Publishing Data Using Java Client Through Thrift/Binary	107
2.3.2.1.4 How to Publish Data Through Other Protocols	116
2.3.2.2 Persisting Data for Batch Analytics	123
2.3.2.3 Configuring DAS to Receive Data	124
2.3.2.3.1 Configuring Event Receivers	124
2.3.2.3.2 Input Mapping Types	161
2.3.2.3.3 Creating Custom Transports to Receive Events	171
2.3.3 Analyzing Data	178
2.3.3.1 Interactive Analytics	178
2.3.3.1.1 Persisting Data for Interactive Analytics	178
2.3.3.1.2 Configuring Indexes	179
2.3.3.1.3 Data Explorer	192
2.3.3.1.4 Activity Explorer	196
2.3.3.1.5 Query Language Reference	199

2.3.3.2 Batch Analytics Using Spark SQL	201
2.3.3.2.1 Batch Analytics Console	202
2.3.3.2.2 Scheduling Batch Analytics Scripts	202
2.3.3.2.3 Spark Query Language	206
2.3.3.2.4 Publishing Events Using Apache Spark	211
2.3.3.2.5 Creating Spark User Defined Functions	213
2.3.3.2.6 Spark Troubleshooting	214
2.3.3.3 Realtime Analytics Using Siddhi	215
2.3.3.3.1 Creating a Standalone Execution Plan	215
2.3.3.3.2 Creating a STORM Based Distributed Execution Plan	218
2.3.3.3.3 Siddhi Query Language	220
2.3.3.3.4 Creating Siddhi Query Templates	276
2.3.3.4 Predictive Analytics Using WSO2 ML	283
2.3.3.4.1 Configuring WSO2 ML with WSO2 DAS	283
2.3.3.4.2 Creating Machine Learning Models	284
2.3.3.4.3 Using a ML Model Within WSO2 CEP	284
2.3.3.4.4 Using a ML Model Within WSO2 ESB	284
2.3.4 Communicating Results	284
2.3.4.1 Visualizing Results	284
2.3.4.1.1 Analytics Dashboard	284
2.3.4.2 Creating Alerts	305
2.3.4.2.1 Event Publisher Types	315
2.3.4.2.2 Output Mapping Types	364
2.3.4.3 Communicating Results Through REST API	370
2.3.4.4 Analytics JavaScript (JS) API	371
2.3.4.4.1 Retrieving the List of Tables of All Record Stores via JS API	372
2.3.4.4.2 Retrieving All Record Stores via JS API	373
2.3.4.4.3 Retrieving the Record Store of a Given Table via JS API	373
2.3.4.4.4 Checking if a Given Table Exists via JS API	374
2.3.4.4.5 Clearing Indexed Data of a Given Table via JS API	374
2.3.4.4.6 Retrieving Records Based on a Time Range via JS API	375
2.3.4.4.7 Retrieving Records Matching the Given Primary Key Combination via JS API	376
2.3.4.4.8 Retrieving Records of Given Record IDs via JS API	377
2.3.4.4.9 Retrieving the Total Record Count of a Table via JS API	378
2.3.4.4.10 Retrieving the Number of Records Matching the Given Search Query via JS API	379
2.3.4.4.11 Retrieving All Records Matching the Given Search Query via JS API	379
2.3.4.4.12 Retrieving the Schema of a Table via JS API	380
2.3.4.4.13 Checking if the Given Records Store Supports Pagination via JS API	381
2.3.4.4.14 Tracking the Indexing Process Completion via JS API	382
2.3.4.4.15 Drilling Down Through Categories via JS API	382
2.3.4.4.16 Retrieving Specific Records through a Drill Down Search via JS API	383
2.3.4.4.17 Retrieving the Number of Records Matching the Drill Down Criteria via JS API	384
2.3.4.4.18 Adding an Event Stream Definition via JS API	385
2.3.4.4.19 Publishing Events to WSO2 DAS via JS API	386
2.3.4.4.20 Retrieving an Existing Event Definition via JS API	387
2.3.4.4.21 Tracking the Indexing Process Completion of a Table via JS API	388
2.3.4.4.22 Retrieving Aggregated Values of Given Records via JS API	388
2.3.5 Packaging Artifacts as a C-App Archive	390

2.3.6 Debugging	399
2.3.6.1 Event Statistics	399
2.3.6.2 Event Tracer	402
2.3.6.3 Siddhi Try It Tool	403
2.4 Admin Guide	405
2.4.1 Spark Configurations	406
2.4.2 Enabling/Disabling selected DAS components	410
2.4.3 Purging Data	412
2.4.4 Analytics Migration Tool	415
2.4.5 Analytics Data Backup / Restore Tool	418
2.4.6 Performance Tuning	422
2.4.7 Configuration Guide	427
2.4.7.1 Registry	427
2.4.7.1.1 Introduction to Registry	427
2.4.7.1.2 Managing the Registry	428
2.4.7.1.3 Searching the Registry	442
2.4.7.1.4 Sharing Registry Space Among Multiple Products	444
2.4.7.2 User Management	463
2.4.7.2.1 Introduction to User Management	464
2.4.7.2.2 Adding and Managing Users and Roles	465
2.4.7.2.3 Realm Configuration	471
2.4.7.2.4 Changing the RDBMS	475
2.4.7.2.5 Configuring Primary User Stores	475
2.4.7.2.6 Configuring Secondary User Stores	492
2.4.7.3 Feature Management	494
2.4.7.3.1 Introduction to Feature Management	494
2.4.7.3.2 Installing and Managing Features	495
2.4.7.3.3 Recovering from Unsuccessful Feature Installation	498
2.4.7.4 Logging	500
2.4.7.5 Datasources	503
2.4.7.5.1 Configuring an RDBMS Datasource	506
2.4.7.5.2 Configuring a Cassandra Datasource	554
2.4.7.5.3 Configuring a HBase Datasource	554
2.4.7.5.4 Configuring a HDFS Datasource	555
2.4.7.5.5 Configuring a Custom Datasource	556
2.4.7.6 Server Roles for C-Apps	558
2.4.7.6.1 Introduction to Server Roles	558
2.4.7.6.2 Adding a Server Role	559
2.4.7.6.3 Deleting a Server Role	560
2.4.7.6.4 Transports	561
2.4.7.7 Scheduling Tasks	577
2.4.7.8 Security	579
2.4.7.8.1 Fixing Security Vulnerabilities	579
2.4.7.8.2 Enabling Java Security Manager	580
2.4.7.8.3 Setting up Keystores	583
2.4.8 Installing Machine Learner Features for the CEP Extension	593
2.5 Samples	597
2.5.1 Sending Notifications Through Published Events Using Spark	598

2.5.2 Analyzing HTTPD Logs	605
2.5.3 Analyzing Smart Home Data	609
2.5.4 Analyzing Realtime Service Statistics	613
2.5.5 Analyzing Wikipedia Data	615
2.6 Reference Guide	622
2.6.1 REST APIs for Analytics Data Service	623
2.6.1.1 Analytics REST API Guide	623
2.6.1.1.1 Checking if a Given Table Exists via REST API	624
2.6.1.1.2 Retrieving the List of Tables of All Record Stores via REST API	625
2.6.1.1.3 Retrieving Records Based on a Time Range via REST API	626
2.6.1.1.4 Retrieving the Total Record Count of a Table via REST API	627
2.6.1.1.5 Retrieving All Records Matching the Given Search Query via REST API	627
2.6.1.1.6 Retrieving the Number of Records Matching the Given Search Query via REST API	628
2.6.1.1.7 Tracking the Indexing Process Completion via REST API	629
2.6.1.1.8 Retrieving the Schema of a Table via REST API	630
2.6.1.1.9 Drilling Down Through Categories via REST API	631
2.6.1.1.10 Retrieving Specific Records through a Drill Down Search via REST API	632
2.6.1.1.11 Retrieving the Number of Records Matching the Drill Down Criteria via REST API	633
2.6.1.1.12 Retrieving Records Matching the Given Primary Key Combination via REST API	634
2.6.1.1.13 Retrieving All Record Stores via REST API	636
2.6.1.1.14 Retrieving the Record Store of a Given Table via REST API	637
2.6.1.1.15 Checking if the Given Records Store Supports Pagination via REST API	638
2.6.1.1.16 Tracking the Indexing Process Completion of a Table via REST API	639
2.6.1.1.17 Retrieving Aggregated Values of Given Records via REST API	639
2.6.1.1.18 Retrieving the Event Count of Range Facets	641
2.6.1.2 CORS Settings for the Analytics REST API	642
2.6.1.3 HTTP Status Codes	642
2.6.2 Default Ports of WSO2 Products	643
2.6.2.1 WSO2 DAS Specific Ports	646
2.6.3 WSO2 Patch Application Process	648
2.6.4 Calling Admin Services from Apps	649
2.6.5 Customizing the Management Console	654
2.6.6 Using Analytics Spark Features in Other WSO2 Products	656
2.6.7 WSO2 DAS Performance Analysis	657
2.7 FAQ	661
2.8 Glossary	663
2.9 Getting Support	664
2.10 Site Map	665

Upgrading from the Previous Release

Given below is how to upgrade from BAM 2.5.0 to DAS 3.0.0. To upgrade from a version older than 2.4.1, start from the doc that was released immediately after your current release and upgrade incrementally.

- [Preparing to upgrade](#)
- [Configuration upgrades](#)

Preparing to upgrade

Download WSO2 DAS 3.0.0.

Configuration upgrades

WSO2 Data Analytics Server Documentation

Welcome to 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.

```

graph LR
    A[Aggregating Data] --> B[Analyzing Data]
    B --> C[Presenting Data]
  
```

Aggregating Data

Analyzing Data

Presenting Data

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 Introduction to WSO2 DAS, its features, architecture, FAQ and how to get help and support.	Getting Started Explains how to download, install, run and get started with WSO2 DAS.	Configuration Guide Explores different options to customize WSO2 DAS according to user-specific needs.
User Guide Explores the 3 main functionalities of WSO2 DAS. Collecting Data Analyzing Data Communicating Results	Admin Guide The following topics explore various product deployment scenarios and other topics useful for system administrators.	Samples Explains several business use cases of WSO2 DAS.

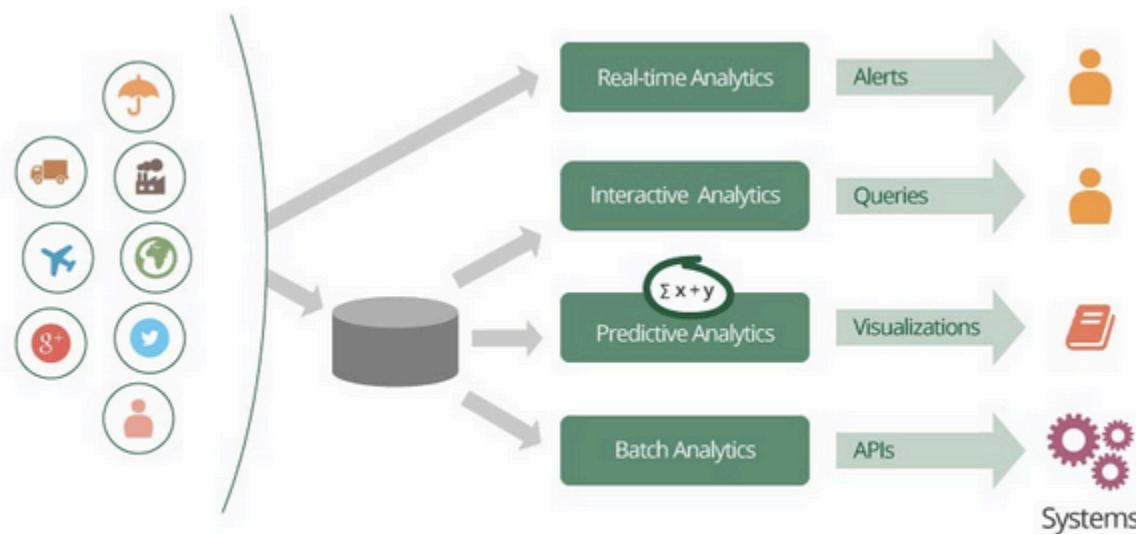
About DAS

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 3.0.0 combines real-time, 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 as illustrated by the image below.



Collect Data → **Analyze & Make Decisions** → **Communicate**

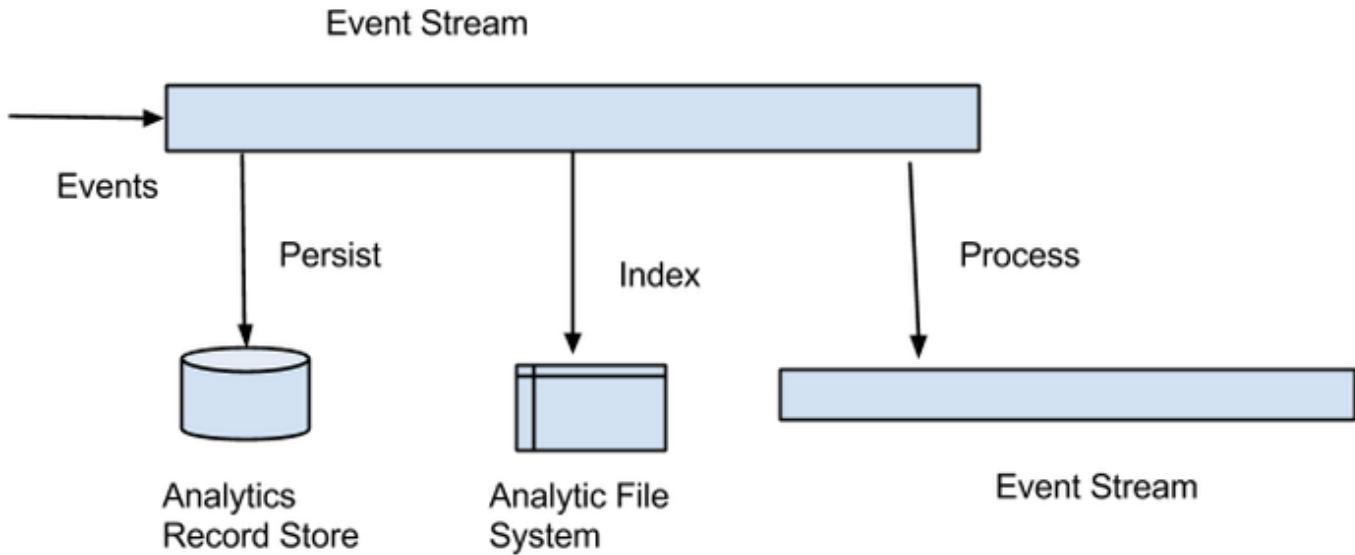
As a part of WSO2's analytics platform, WSO2 DAS introduces a single solution with the ability to build systems and applications that collect and analyze both realtime and persisted, data and communicate the results. It is designed to treat millions of events per second, and is therefore capable to handle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

WSO2 DAS 3.0.0 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 below 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

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 or batch and/or real time analytics.

Batch 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 need to be executed. For more information, see [Batch Analytics Using Spark SQL](#).

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](#).

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](#)

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](#).

Interactive analytics

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.

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](#).

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

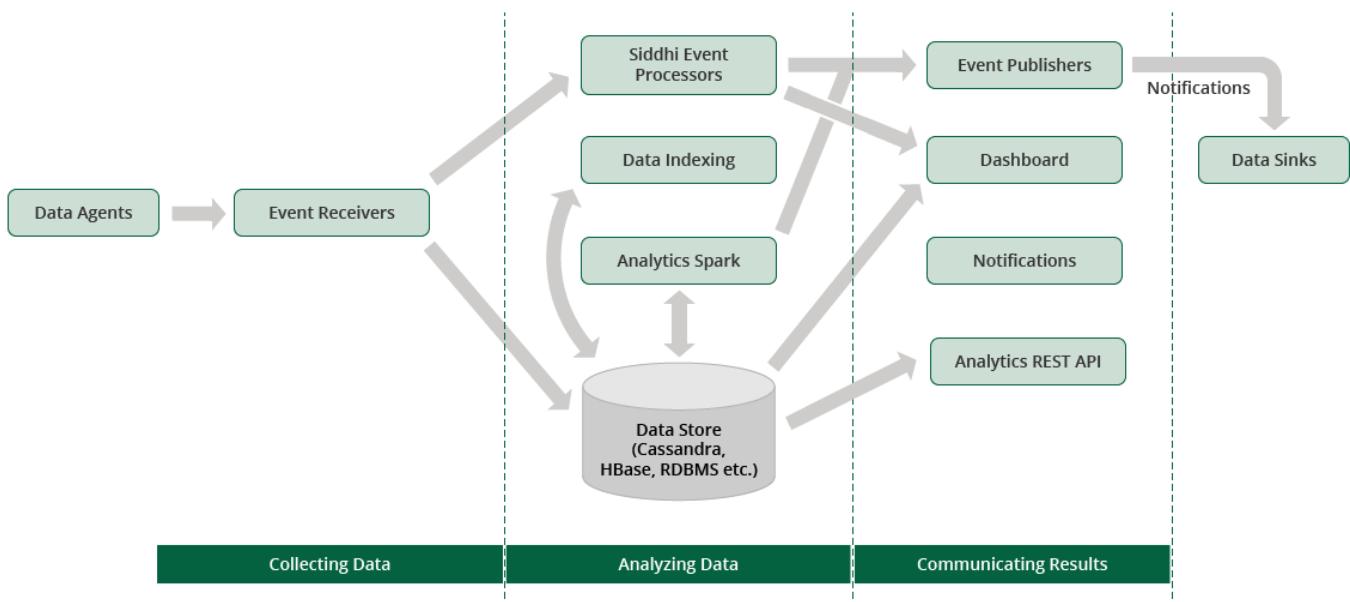
Feature	Description
Data aggregation	<ul style="list-style-type: none"> • Receives data from event sources through Java agents (Thrift, Kafka, JMS), JavaScript clients (Web Sockets, REST), to IoT (MQTT), and also from WSO2 Enterprise Service Bus Connectors. • Publishes events to one API for real-time, batch or interactive processing. • Ability to access the analytics service via comprehensive REST API.
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.
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.

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 be 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](#)
- [Analytics Spark](#)
- [Data Indexing](#)
- [Siddhi Event Processors](#)
- [Event Publishers](#)
- [Analytics Dashboard](#)
- [Event Sinks](#)

Data Agents

Data Agents are external sources which publishes data to WSO2 DAS. Data Agent components reside in external systems and pushes 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](#).

Analytics Spark

Main analytics engine of WSO2 DAS is based on [Apache Spark](#). This 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 [Data Analysis](#).

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.

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 [Working with Execution Plans](#).

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 [Publishing Events](#).

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.

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

What is new in this release

The WSO2 DAS version **3.0.0** is the successor of WSO2 BAM, and it is a complete rewrite of the analytics solution based on the latest technologies. Some of the prominent features and enhancements are as follows.

- Supports batch processing with Apache Spark (SQL).
- Supports distributed data indexing.
- Pluggable data sources support with the new data abstraction layer
- Unified data querying with Analytics REST API

Also, WSO2 DAS contains the following major new technological changes/substitutions when compared to its predecessor.

- Introduction of a generic data store that can mount RDBMS, HBase, Cassandra or any other data store, instead of supporting Cassandra as the one and only event store.
- Replaced Hadoop with Apache Spark, and Hive with Spark SQL.
- Introduced indexing on stream persistence based on Apache Lucene, instead of the Casandra secondary and custom index based indexing.
- Replaced the Gadget Server and Gadget generating tool with WSO2 UES-based dashboards and its new gadget generating tool.
- Integration of WSO2 CEP 4.0.0 based features, instead of WSO2 CEP 3.x.
- Introduced CAR file-based artifact deployment for WSO2 BAM toolbox support.

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 3.0.0.

- APIM 1.9.1
- AS 5.3.0
- ESB 4.9.0

Fixed issues

For a list of fixed issues in this release, see [WSO2 DAS 3.0.0 - Fixed Issues](#).

Known issues

For a list of known issues in this release, see [WSO2 DAS 3.0.0 - 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 BAM 2.5.0
- Running the Product
- Get Involved
- WSO2 DAS Production Setup Checklist

[Go to Home Page](#)

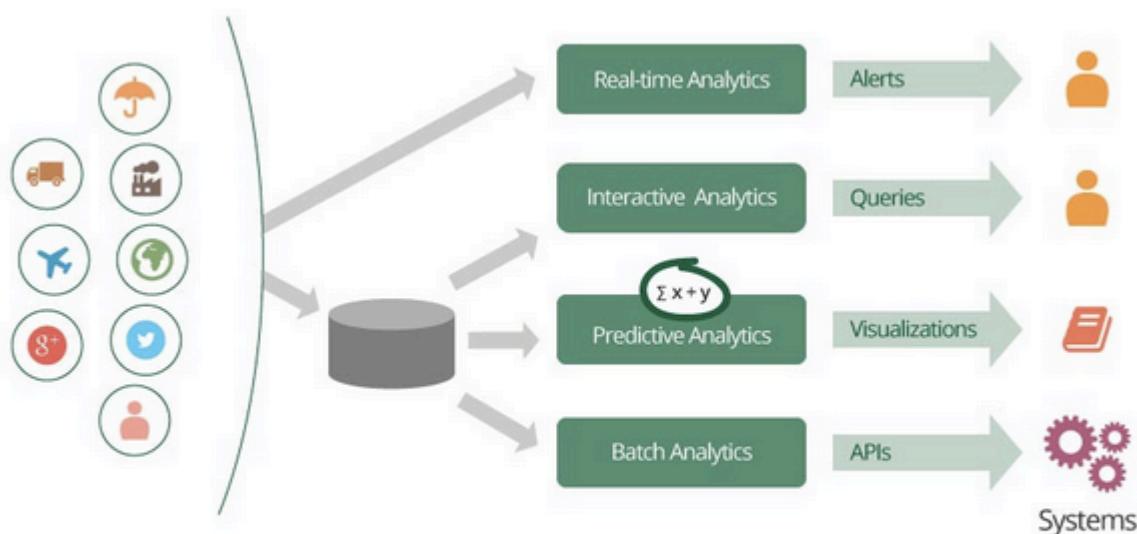
Quick Start Guide

This guide provides a quick introduction to using WSO2 Data Analytics Server (DAS).

- About WSO2 DAS
- About this guide
- Getting started
- Deploying the sample C-App
- WSO2 DAS basics
- Where to go next

About WSO2 DAS

WSO2 Data Analytics Server 3.0.0 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 combines real-time, 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.



Collect Data

Analyze & Make Decisions

Communicate

About this guide

This introductory guide demonstrates the steps required to get a simple scenario working based on the workflow of WSO2 DAS . In this guide, a collection of events (a CSV file) which contains a set of records collected from smart plug sensors in households is used to publish (simulate) events to WSO2 DAS for data collection. Thereby, this guide demonstrates calculating plug usage per household on the collected data for batch analytics, calculating the average, minimum, and maximum values for the data inflow for realtime analytics, and performing an ad hoc Apache Lucene query on the data for interactive analytics. Further, it communicates the visualization of results through dashboards.

Getting started

Set up the following prerequisites before you begin.

1. Set up the appropriate general prerequisite applications before you start. For information on the general prerequisites, see [Installation Prerequisites](#).
2. Download WSO2 Data Analytics Server. For instructions, see [Downloading the Product](#).
3. Install the product by setting the `JAVA_HOME` environment variable and other system properties. For instructions, see [Installing the Product](#).
4. Start the DAS by navigating to `<DAS_HOME>/bin/` using the command-line, and executing `wso2server.bat` (for Windows) or `wso2server.sh` (for Linux). For instructions, see [Running the Product](#).

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 [Carbon Application Deployment for DAS](#). 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>/samples/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.

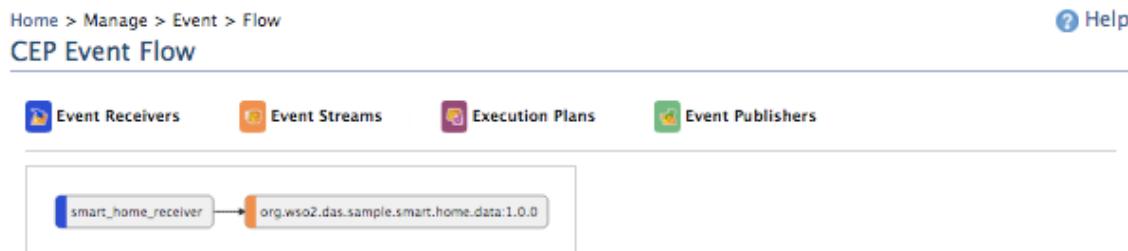
Home > Manage > Carbon Applications > List

Carbon Applications List

1 Running Carbon Applications.

Carbon Applications	Version	Actions
Smart_Home_Sample_CApp	1.0.0	Delete Download

-  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.

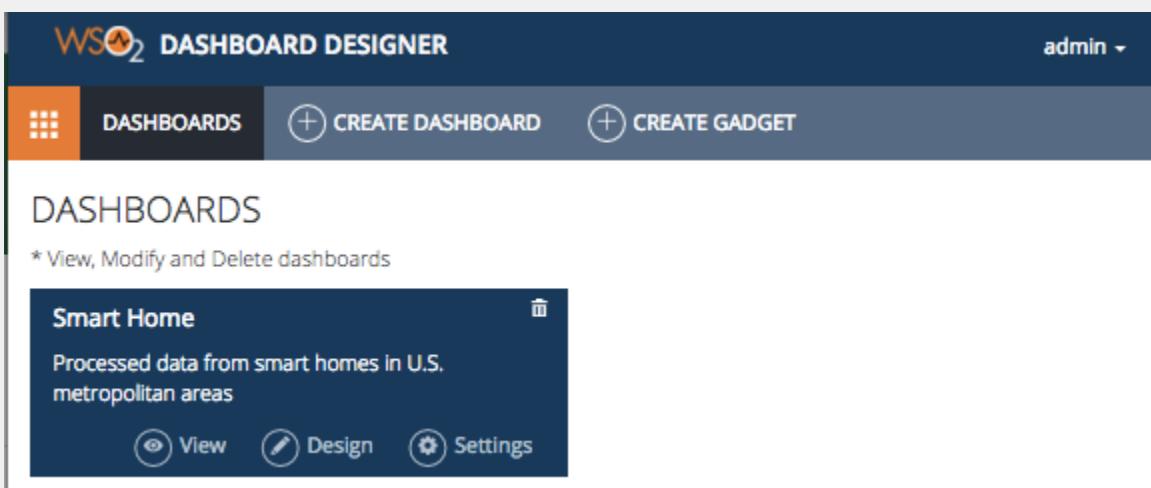
Navigate to `<DAS_HOME>/samples/smart-home/` directory in a new CLI tab, and execute the following command to run the data publisher: `ant`

-  This 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 `Smart_Home.car` file.

Viewing the output

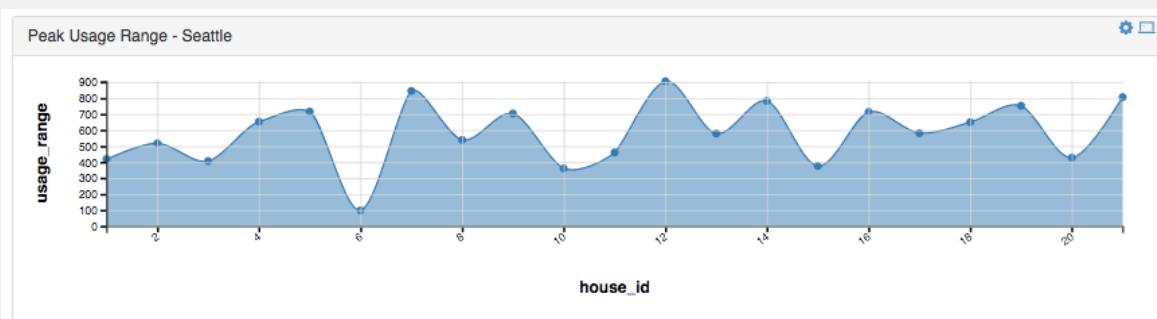
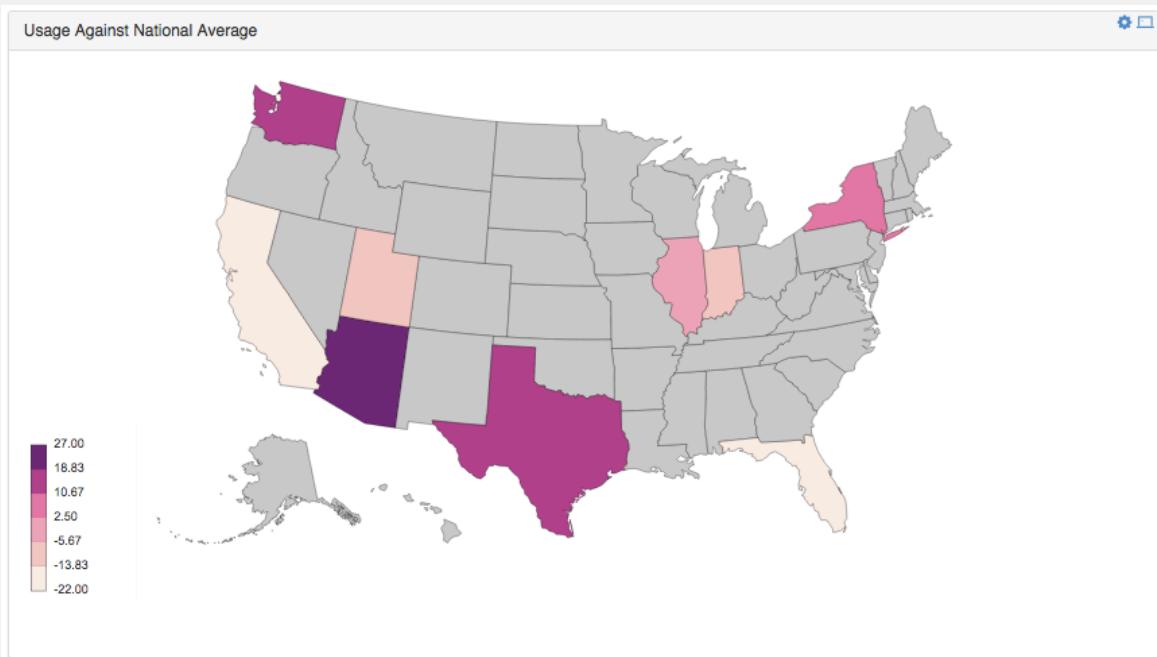
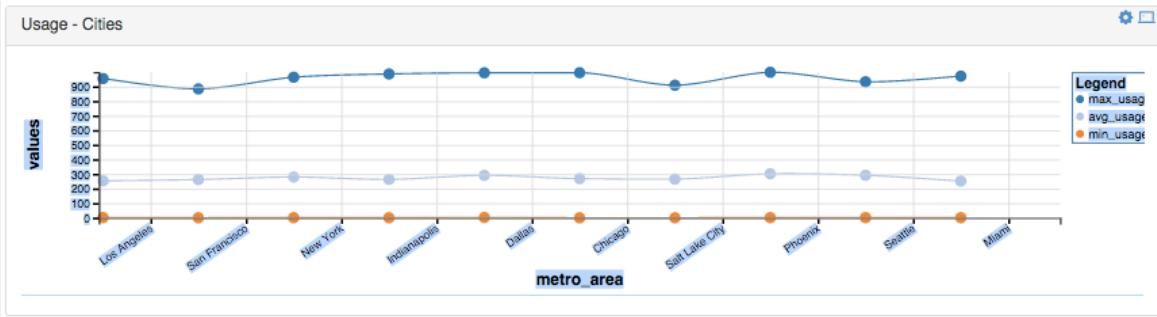
Follow the steps below to view the presentation of the output in the Analytics Dashboard.

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. You view the dashboard deployed by the C-App as shown below.



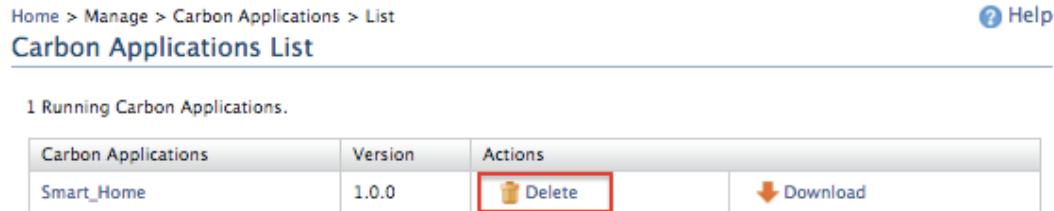
The screenshot shows the 'WSO2 DASHBOARD DESIGNER' interface. The top navigation bar includes 'admin ▾' and tabs for 'DASHBOARDS', 'CREATE DASHBOARD', and 'CREATE GADGET'. Below the header, the main area is titled 'DASHBOARDS' with a sub-instruction: '* View, Modify and Delete dashboards'. A single dashboard card is visible, titled 'Smart Home' with the subtext 'Processed data from smart homes in U.S. metropolitan areas'. At the bottom of the card are three buttons: 'View', 'Design', and 'Settings'.

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.



The screenshot shows the 'Carbon Applications List' page. At the top, there is a breadcrumb navigation: Home > Manage > Carbon Applications > List. On the right, there is a 'Help' link. Below the header, it says '1 Running Carbon Applications.' A table is displayed with the following data:

Carbon Applications	Version	Actions
Smart_Home	1.0.0	 Delete 

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.

WSO2 DAS basics

The following sections provide detailed instructions on the main functionalities of WSO2 DAS.

- Collecting data
- Analysing data
- Communicating results

You can perform data collection in WSO2 DAS as described in the next [Collecting data](#) section below.

Collecting data

The first step of WSO2 DAS workflow is to collect data. In the data collection process, first you need to create the event stream definition. An event is a unit of data collection, and an event stream is a sequence of events of a particular type which consists of a set of unique attributes.

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

Creating the event stream

The first step in collecting data is defining an event stream by creating the event stream definition. The defined stream provides the structure required to process the events. For more information on event streams, see [Event Streams](#). Follow the steps below to create the event stream.

1. Log in to the DAS Management Console using the following URL and admin/admin credentials:
<https://10.100.5.72:9443/carbon/>
2. Click **Main**, and then click **Streams**.

3. Click **Add Event Stream**, and enter the details as shown below.

Event Stream Details

Attribute Name	Attribute Type	Actions
<code>id</code>	string	
<code>value</code>	float	
<code>property</code>	bool	
<code>plug_id</code>	int	
<code>household_id</code>	int	
<code>house_id</code>	int	

Event Stream Details

Parameter	Value
Event Stream Name	SMARTHOME_DATA
Event Stream Version	1.0.0

Payload Data Attributes

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

Attribute	Attribute Type
<code>id</code>	string
<code>value</code>	float
<code>property</code>	bool
<code>plug_id</code>	int
<code>household_id</code>	int
<code>house_id</code>	int

4. Click **Add Event Stream**. The new event stream is added to the list of all available event streams as shown below.

Available Event Streams		
Add Event Stream		
1 Event streams available		
Event Stream Id	Event Stream Description	Actions
SMARTHOME_DATA:1.0.0		Simulate Delete Edit

Persisting the event stream

Events received by the DAS can be processed either in realtime and/or in batch mode. You need to persist the event information, to process the events in batch mode. However, if you process the events in realtime you do not need to persist them. For persisted events, configurable options are provided to index the data.

WSO2 DAS introduces a pluggable architecture which allows you to persist data events into any Relational Data Storage (Oracle, MSSQL, MySQL etc.) or NoSQL storages (Apache HBase and Apache Cassandra). Multi data event storage is also possible. For example, events can be stored in a NoSQL storage while the processed data events can be stored in a relational data storage.

Follow the steps below to persist received events.

- For persisted events, configurable options are provided to index the data which is required for Interactive Analytics later in the example.
- Log in to the DAS Management Console, if you are not already logged in.
- Click **Main**, and then click **Streams**.
- Click **Edit** option of the corresponding event stream as shown below.

Home > Manage > Event > Streams

Available Event Streams

+ Add Event Stream

1 Event streams available

Event Stream Id	Event Stream Description	Actions
SMARTHOME_DATA:1.0.0		

- Click **Next[Persist Event]**.
- Select the **Persist Event Stream** check box. As a result, the **Persist Attribute** check boxes are selected for all the attributes. Then select the **Index Column** checkbox for the `house_id` attribute as shown below.

Enter Event Stream Details

Persist Event Stream

Record Store
EVENT_STORE

Payload Data Attributes

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
<input checked="" type="checkbox"/>	id	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	value	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	property	BOOLEAN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	plug_id	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	household_id	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	house_id	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Arbitrary Data Attributes

No arbitrary data attributes are defined

Attribute Name : Attribute Type : INTEGER Primary Key : Index Column : Score Param : Add

Back Save Event Stream

- Click **Save Event Stream**.
- Click **Yes** in the pop-up message as shown below.

WSO2 Carbon

If event stream is edited then related configuration files will be also affected! Are you sure want to edit?

Yes **No**

- You view the persisted event stream added to the list of all available event streams as shown below.

Home > Manage > Event > Streams

Available Event Streams

+ Add Event Stream

1 Event streams available

Event Stream Id	Event Stream Description	Actions
SMARTHOME_DATA:1.0.0		

Creating the event receiver

Once you define the event stream and configure how it should be used, you need to create event receivers to connect WSO2 DAS with different data sources.

WSO2 DAS supports event retrieval from many transport protocols and different formats. For information on supported transport protocols and type formats, see [Configuring Event Receivers](#). Follow the steps below to create an event receiver of the `WSO2Event` type for this guide.

i WSO2Event event receiver is used to receive events in the WSO2Event format via Thrift or binary protocols. For more information, see [WSO2Event Event Receiver](#).

1. Log in to the DAS Management Console, if you are not already logged in.
2. Click **Main**, and then click **Receivers**.
3. Click **Add Event Receiver**, and enter the details as shown below.

Create a New Event Receiver

Enter Event Receiver Details	
Event Receiver Name*	DATA_RECEIVER <small>Enter a unique name to identify Event Receiver</small>
From	
Input Event Adapter Type*	wso2event <small>Select the type of Adapter to receive events</small>
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>}, ...}	
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	false <small>This depends on how events are published to the server, 'true' only if multiple receiver URLs are defined in different receiver groups ({}).</small>
To	
Event Stream*	SMARTHOME_DATA:1.0.0 <small>The event stream that will be generated by the received events</small>
Mapping Configuration	
Message Format*	wso2event <small>Select the input message format</small>
Advanced	
<input type="button" value="Add Event Receiver"/>	

Parameter	Value
Event Receiver Name	DATA_RECEIVER
Input Event Adapter Type	wso2event
Event Stream	SMARTHOME_DATA:1.0.0
Message Format	wso2event

4. Click **Add Event Receiver**. You view the new event receiver added to the list of all available event receivers as shown below.

Home > Manage > Event > Receivers

Available Event Receivers

Available Event Receivers				
Add Event Receiver <small>1 Active Event Receivers. 0 Inactive Event Receivers (All)</small>				
Event Receiver Name	Message Format	Input Event Adapter Type	Input Stream ID	Actions
DATA_RECEIVER	wso2event	wso2event	SMARTHOME_DATA:1.0.0	Enable Statistics Enable Tracing Delete Edit

Creating another event stream

The SMARTHOME_DATA event stream you have already created serves as the input stream in this scenario. Events of this stream need to be forwarded to another stream once they are processed in order to be published. Follow the steps below to add the output event stream to which the processed data is forwarded.

1. Log in to the DAS Management Console if you are not already logged in.
2. Click **Main**, and then click **Streams**.
3. Click **Add Event Stream**, and enter the details as shown below.

Home > Manage > Event > Streams Help

Define New Event Stream

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

Event Stream Name*	<input type="text" value="usageStream"/> <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																					
Meta Data Attributes <i>No meta data attributes are defined</i>																					
Attribute Name :	<input type="text"/>	Attribute Type :	<input type="button" value="int"/> <input type="button" value="Add"/>																		
Correlation Data Attributes <i>No correlation data attributes are defined</i>																					
Attribute Name :	<input type="text"/>	Attribute Type :	<input type="button" value="int"/> <input type="button" value="Add"/>																		
Payload Data Attributes <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Attribute Name</th> <th>Attribute Type</th> <th>Actions</th> </tr> </thead> <tbody> <tr> <td>house_id</td> <td>int</td> <td><input type="button" value="Delete"/></td> </tr> <tr> <td>maxVal</td> <td>float</td> <td><input type="button" value="Delete"/></td> </tr> <tr> <td>minVal</td> <td>float</td> <td><input type="button" value="Delete"/></td> </tr> <tr> <td>avgVal</td> <td>double</td> <td><input type="button" value="Delete"/></td> </tr> <tr> <td>currentTime</td> <td>string</td> <td><input type="button" value="Delete"/></td> </tr> </tbody> </table>				Attribute Name	Attribute Type	Actions	house_id	int	<input type="button" value="Delete"/>	maxVal	float	<input type="button" value="Delete"/>	minVal	float	<input type="button" value="Delete"/>	avgVal	double	<input type="button" value="Delete"/>	currentTime	string	<input type="button" value="Delete"/>
Attribute Name	Attribute Type	Actions																			
house_id	int	<input type="button" value="Delete"/>																			
maxVal	float	<input type="button" value="Delete"/>																			
minVal	float	<input type="button" value="Delete"/>																			
avgVal	double	<input type="button" value="Delete"/>																			
currentTime	string	<input type="button" value="Delete"/>																			
Attribute Name :	<input type="text"/>	Attribute Type :	<input type="button" value="int"/> <input type="button" value="Add"/>																		
Add Event Stream Next [Persist Event]																					

Event Stream Details

Parameter	Value
Event Stream Name	usageStream
Event Stream Version	1.0.0

Payload Data Attributes

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

Attribute	Attribute Type
house_id	int
maxVal	float
minVal	float
avgVal	double
currentTime	string

4. Click **Next[Persist Event]**.
5. Select the **Persist Event Stream** check box. As a result, the **Persist Attribute** check box is selected for all the attributes as shown below.

The screenshot shows the 'Create Stream Definition' dialog with the title 'Define New Event Stream'. Under 'Enter Event Index Details', the 'Persist Event Stream' checkbox is checked. In the 'Record Store' dropdown, 'EVENT_STORE' is selected. The 'Payload Data Attributes' section contains four entries: 'house_id' (Attribute Type: INTEGER), 'maxVal' (Attribute Type: FLOAT), 'minVal' (Attribute Type: FLOAT), and 'avgVal' (Attribute Type: DOUBLE). All four attributes have their 'Persist Attribute' checkboxes checked. Below this, there is a section for 'Arbitrary Data Attributes' which is currently empty. At the bottom, there are buttons for 'Back', 'Save Event Stream', and 'Add'.

6. Click **Add Event Stream**. The new event stream is added to the list of all available event streams as shown below.

The screenshot shows the 'Available Event Streams' page. At the top, there is a breadcrumb navigation: Home > Manage > Event > Streams. On the right, there is a 'Help' link. The main area is titled 'Available Event Streams' and shows a list of event streams. There is a green '+' button labeled 'Add Event Stream'. Below it, it says '2 Event streams available'. The table lists two entries:

Event Stream Id	Event Stream Description	Actions
SMARTHOME_DATA:1.0.0		
usageStream:1.0.0		

Creating an event publisher

Once the events are processed, events publishers are used to publish results to external systems for taking further actions. Event publishers provide the capability to send event notifications and alerts from WSO2 DAS to external systems. Follow the steps below to create a new event publisher.

1. Log in to the DAS Management Console, if you are not already logged in.
2. Click **Main**, and then click **Publishers** in the **Event** menu.
3. Click **Add Event Publisher**, and enter the following details as shown below.

[Create a New Event Publisher](#)

Enter Event Publisher Details	
Event Publisher Name*	DATA_PUBLISHER <small>Enter a unique name to identify Event Publisher</small>
From	
Event Source*	usageStream:1.0.0 <small>The stream of events that need to be published</small>
Stream Attributes	house_id int, maxVal float, minVal float, avgVal double
To	
Output Event Adapter Type*	logger <small>Select the type of Adapter to publish events</small>
<i>Dynamic Adapter Properties</i>	
Unique Identifier	<small>To uniquely identify a log entry</small>
<i>Mapping Configuration</i>	
Message Format*	text <small>Select the output message format</small>
Advanced	
<input type="button" value="Add Event Publisher"/> <input type="button" value="Test Event Publisher"/>	

Parameter	Value
Event Publisher Name	DATA_PUBLISHER
Event Source	usageStream:1.0.0
Output Event Adapter Type	logger
Message Format	text

4. Click **Add Event Publisher**. You view the new event publisher added to the list of all event publishers as shown below.

i Since you created a logger type event publisher, the output is written to the CLI.

Home > Manage > Event > Publishers

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
DATA_PUBLISHER	text	logger	usageStream:1.0.0	Enable Statistics Enable Tracing Delete Edit

Publishing events

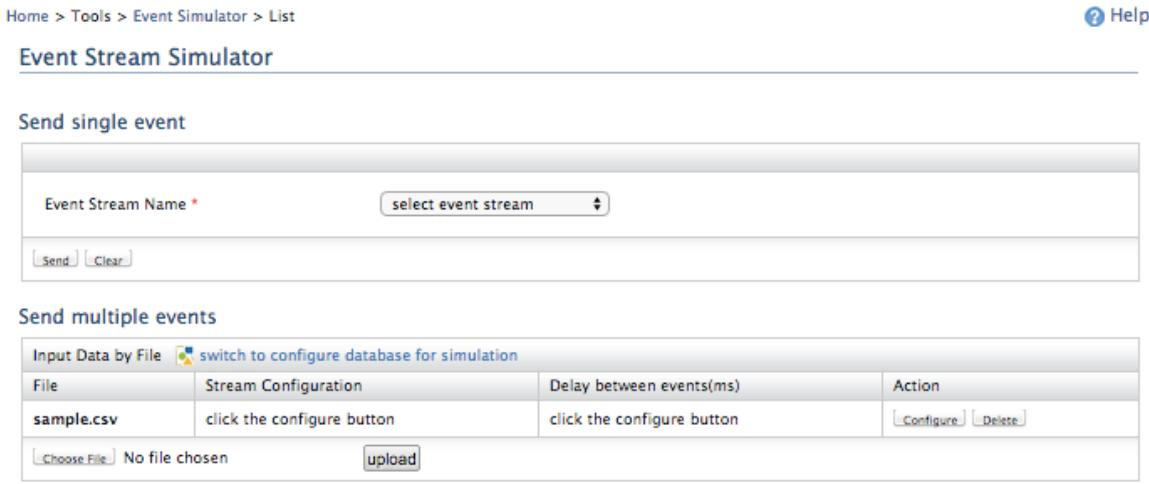
Event Simulator is a tool which you can use for publishing events to event streams. You need to create event(s) by assigning values to event stream attributes to simulate them. For more information, see [Publishing Data Using Event Simulation](#). Follow the steps below to perform event simulation.

1. Log in to the DAS Management Console, if you are not already logged in.
2. Click **Tools**, and then click **Event Simulator**.

3. Download the `sample.csv` file which contains a set of events records collected from household 'smart plug' sensors.

 If you have the `sample.csv` file already uploaded you can skip step 4 to 6 below.

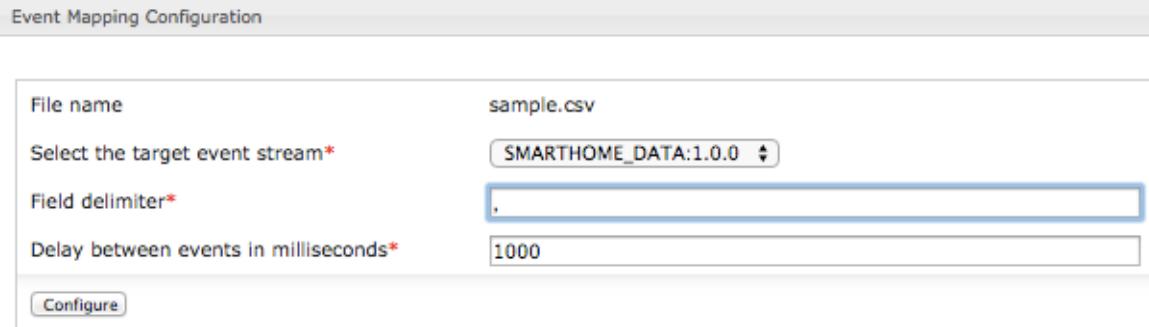
4. In the **Send multiple events** option, click **Choose File**.
5. Select the `sample.csv` file which you downloaded, and click **Upload**.
6. Click **OK** in the pop up message which indicates successful uploading of the CSV file, and refresh the page to view the uploaded file which is displayed as shown below.



The screenshot shows the 'Event Stream Simulator' interface. Under 'Send single event', there is a dropdown menu labeled 'select event stream'. Under 'Send multiple events', there is a table with columns: 'File', 'Stream Configuration', 'Delay between events(ms)', and 'Action'. A row is selected with 'sample.csv' in the 'File' column, 'click the configure button' in 'Stream Configuration', and 'click the configure button' in 'Delay between events(ms)'. The 'Action' column contains 'Configure' and 'Delete' buttons. Below the table is a file input field with 'No file chosen' and an 'upload' button.

7. Click **Configure**, and enter the details as shown below.

-  Select `SMARTHOME_DATA:1.0.0` for Select the target event stream, and type a comma in the provided text field for **Field delimiter**.



The screenshot shows the 'Event Mapping Configuration' dialog. It has fields for 'File name' (set to 'sample.csv'), 'Select the target event stream*' (set to 'SMARTHOME_DATA:1.0.0'), 'Field delimiter*' (set to ','), and 'Delay between events in milliseconds*' (set to '1000'). At the bottom is a 'Configure' button.

Parameter	Value
File Name	sample.csv
Select the target event stream	SMARTHOME_DATA:1.0.0
Field delimiter	,
Delay between events in milliseconds	1000

8. Click **Configure**, and then click **OK** in the message which pops up.

9. Play to start simulating the events in the uploaded file to publish events as shown below.

Send multiple events			
Input Data by File switch to configure database for simulation			
File	Stream Configuration	Delay between events(ms)	Action
sample.csv	SMARTHOME_DATA:1.0.0	1000	Play Configure Delete
<input type="button" value="Choose File"/> No file chosen		<input type="button" value="upload"/>	<input type="button" value="cancel"/>

You can analyse data received by WSO2 DAS as described in the next [Analysing data](#) section below.

Analysing data

You can configure any data event stream received by WSO2 DAS to perform batch, real time, and/or interactive analytics as described in the below sections. The first section demonstrates how to perform batch analytics.

- [Batch analytics](#)
- [Realtime analytics](#)
- [Interactive analytics](#)

Batch 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. For more information, see [Data Analysis](#). You can perform batch analytics either [using Spark analytics scripts](#) or [using the Spark Console](#) as described below.

- You need to follow instructions in the [Collecting data](#) section (i.e. create the event stream, persist it, create the event receiver, and publish events to WSO2 DAS), before performing the following batch analytics operations.

Using the analytics script

Follow the steps below to create the analytics script.

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.
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 homeData USING CarbonAnalytics OPTIONS
(tableName "SMARTHOME_DATA", schema "id STRING, value FLOAT,
property BOOLEAN, plug_id INT, household_id INT, house_id
INT" );

create temporary table plugUsage using CarbonAnalytics options
(tableName "plug_usage", schema "house_id INT, household_id
INT, plug_id INT, usage FLOAT -sp");

insert overwrite table plugUsage select house_id, household_id,
plug_id, max(value) - min (value) as usage from homeData where
property = false group by house_id, household_id, plug_id ;

Select * from plugUsage where usage>300

```

i 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 calculating the usage value (through the difference of the max and min values), grouped by `house_id`, `household_id`, and `plug_id` from data of the temporary `homeData` table.
- Writes back to a new DAL table named `plug_usage`.
- Executes the following query: `Select * from plugUsage where usage>300`

5. Click **Execute Script**, to check the validity of the script as shown below. You view the results as shown below.

house_id	household_id	plug_id	usage
0	1	5	449.576

1 rows returned.

6. Click **Add**. You view the new script added to the list of all available scripts as shown below.

Available Analytics Scripts	
Scripts	Actions
BATCH_ANALYTICS_SCRIPT	Edit Execute Delete
+ Add New Analytics Script	

Using the Spark Console

You can obtain faster results by executing adhoc queries on the indexed attributes through an interactive Web console, named as the named as the **Spark Console**. Follow the steps below to perform a batch analytics operation using the Spark Console.

1. Log in to the DAS Management Console, if you are not already logged in.
2. Click **Main**, and then click **Console** in the **Batch Analytics** menu.
3. Enter the following Spark SQL query in the console, and press **Enter** key.

```
Select * from plugUsage where usage>100
```

You view the output as shown below.

SparkSQL> select * from plugUsage where usage > 100; Show 10 entries			
house_id	household_id	plug_id	usage
1	2	5	209.877
2	5	3	196.72
2	5	7	353.636
2	2	3	201.472
2	2	4	210.61101
2	2	5	193.173
3	0	9	246.91801
3	0	13	354.94598
3	1	0	228.89
4	5	5	121.12401

Showing 21 to 30 of 228 entries

You can perform realtime analytics in WSO2 DAS as described in the next Realtime analytics section below .

Realtime analytics

The realtime analytics engine uses a set of specified queries or rules through a SQL-like Siddhi Query Language defined in an execution plan, to process multiple event streams in realtime. An execution plan consists of a set of queries and import and export streams. It is the store of the event logic that is bound to an instance of the server runtime, and acts as the editor for defining the event processing logic. For more information see, [Working with Execution Plans](#) . You can perform for realtime analytics using the same event stream which you used before to perform batch analytics by proceeding the event streams inflow through the WSO2 real time analytics engine as described below.

- ✓ You need to follow instructions in the [Collecting data](#) section (i.e. create the event stream, persist it, create the event receiver, create the event publisher, and publish events to WSO2 DAS), before performing the following realtime analytics operations.

Creating the execution plan

Follow the steps below to create an execution plan.

1. Log in to the DAS Management Console, if you are not already logged in.
2. Click **Main**, and then click **Execution Plans** in the **Streaming Analytics** menu.
3. Click **Add Execution Plan**, and enter the following details as shown below.

i In the below execution plan, the realtime engine (Siddhi) collects 10 incoming data events within a one minute time window from the incoming filtered stream, calculates the average, maximum and minimum values in realtime grouped by house_id, and sends the processed event to another newly defined stream.

- Select `SMARTHOME_DATA:1.0.0` for **Import Stream**, enter `inputStream` for **As**, and click **Import**.
- Add the following query at the end of the provided space.

```
from inputStream[value>0]#window.time(1 min)
select house_id,max(value) as maxVal,min(value) as
minVal, avg(value) as avgVal,time:currentTime() as
currentTime
group by house_id
insert current events into usageStream ;
```

Create a New Execution Plan

Enter Event Processor Details	
Query Expressions	
Import Stream*	Import Stream : <code>SMARTHOME_DATA:1.0.0</code> As : <code>inputStream</code> <input type="button" value="Import"/>
Export Stream	Value Of : <code>usageStream</code> StreamId : <code>-- Create Stream Definition --</code> <input type="button" value="Export"/>
<pre>1 /* Enter a unique ExecutionPlan */ 2 @Plan:name('ExecutionPlan') 3 4 /* Enter unique description for ExecutionPlan */ 5 -- @Plan:description('ExecutionPlan') 6 7 /* define streams/tables and write queries here ... */ 8 9 @Import('SMARTHOME_DATA:1.0.0') 10 define stream inputStream(id string, value float, property bool, plug_id int, household_id int, house_id int); 11 12 @Export('usageStream:1.0.0') 13 define stream usageStream(house_id int, maxVal float, minVal float, avgVal double,currentTime string); 14 15 from inputStream[value>0]#window.time(1 min) 16 select house_id,max(value) as maxVal,min(value) as minVal, avg(value) as avgVal,time:currentTime() as currentTime 17 group by house_id 18 insert current events into usageStream ;</pre>	
<input type="button" value="Validate Query Expressions"/>	
<input type="button" value="Add Execution Plan"/>	

4. Select usageStream 1.0.0 for **StreamID of Export Stream**, and click **Export** as shown below.

```

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('SMARTHOME_DATA:1.0.0')
10 define stream inputStream (id string, value float, property bool, plug_id int, household_id int, house_id int);
11 #Export('usageStream:1.0.0')
12 define stream usageStream (house_id int, maxVal float, minVal float, avgVal double, currentTime string);
13
14 from inputStream[value>0]#window.time(1 min)
15 select house_id,max(value) as maxVal,min(value) as minVal, avg(value) as avgVal,time:currentTime() as currentTime
16 group by house_id
17 insert current events into usageStream ;

```

Validate Query Expressions

Add Execution Plan

5. Click **Validate Query Expressions**, to validate the execution plan.
 6. Click **Add Execution Plan**. You view the new execution plan added to the list of all available execution plans as shown below.

Execution Plan Name	Description	Actions
ExecutionPlan		

7. Publish events to WSO2 DAS by simulating events. For instructions, see [Publishing events](#).
 8. View the output logged with the published events in the CLI on which you ran WSO2 DAS as shown below.

```

currentTime:11:59:16
[2015-08-26 11:59:18,143] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: DATA_PUBLISHER,
Event: house_id:39,
maxVal:252.957,
minVal:0.242,
avgVal:31.40709863515401,
currentTime:11:59:18
[2015-08-26 11:59:20,145] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: DATA_PUBLISHER,
Event: house_id:39,
maxVal:252.957,
minVal:0.242,
avgVal:32.82742822241215,
currentTime:11:59:20
[2015-08-26 11:59:22,147] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: DATA_PUBLISHER,
Event: house_id:39,
maxVal:252.957,
minVal:0.242,
avgVal:31.5783633000000488,
currentTime:11:59:22
[2015-08-26 11:59:24,150] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: DATA_PUBLISHER,
Event: house_id:39,
maxVal:252.957,
minVal:0.242,
avgVal:33.760695245602854,
currentTime:11:59:24
[2015-08-26 11:59:25,150] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: DATA_PUBLISHER,
Event: house_id:39,
maxVal:252.957,
minVal:0.242,
avgVal:33.30037458303074,
currentTime:11:59:25
[2015-08-26 11:59:26,150] INFO {org.wso2.carbon.event.output.adapter.logger.LoggerEventAdapter} - Unique ID: DATA_PUBLISHER,
Event: house_id:39,
maxVal:252.957,
minVal:0.242,
avgVal:41.425919414162635,
currentTime:11:59:26

```

You can perform interactive analytics in WSO2 DAS as described in the next [Interactive analytics](#) section below .

Interactive analytics

Interactive analytics are used for retrieval of fast results through ad hoc querying of a received and processed data. It is possible in the DAS, when you select to index event stream attributes. You can obtain faster results by executing ad hoc queries on the indexed attributes through the provided Data Explorer.



- You need to follow instructions in the [Collecting data](#) section (i.e. create the event stream, persist it, create the event receiver, and publish events to WSO2 DAS), before performing the following interactive analytics operations. The house_id field which you indexed in the [step 5 of data collection](#) is used to search data that match the specified query.

Using the Data Explorer

The [Data Explorer](#) is the Web console for searching analytical data. Primary key, data range, facet search options are available for simple analytical record searches. It is also possible to search records by providing Lucene queries for advanced searches. Follow the steps below to perform an interactive analytics operation using the [Data Explorer](#).

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 SMARTHOME_DATA for the **Table Name** parameter.
4. Select **By Query** option and enter house_id:39 in the search field as shown below.

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

Data Explorer

Search

Table Name* SMARTHOME_DATA Schedule Data Purging

Search By Date Range By Query

house_id:39

Search Reset

5. Click **Search**. You view the output as shown below.

Results

SMARTHOME_DATA

	id	value	property	plug_id	household_id	house_id	timestamp
1	3655508736	17.984	False	1	12	39	2015-08-26 14:42:36 IST
2	3655508726	0.0	False	12	11	39	2015-08-26 14:42:26 IST
3	3655508722	49.358	False	1	11	39	2015-08-26 14:42:22 IST
4	3655508738	0.0	False	2	12	39	2015-08-26 14:42:38 IST
5	3655508731	3.484	False	5	11	39	2015-08-26 14:42:31 IST
6	3655508729	0.242	False	2	11	39	2015-08-26 14:42:29 IST
7	3655508732	0.0	False	5	11	39	2015-08-26 14:42:32 IST
8	3655508737	3.025	False	2	12	39	2015-08-26 14:42:37 IST
9	3655508735	79.729	False	1	12	39	2015-08-26 14:42:35 IST
10	3655508725	4.087	False	12	11	39	2015-08-26 14:42:25 IST
11	3655508728	40.791	False	13	11	39	2015-08-26 14:42:28 IST
12	3655508724	19.148	False	11	11	39	2015-08-26 14:42:24 IST
13	3655508727	94.326	False	13	11	39	2015-08-26 14:42:27 IST
14	3655508734	0.0	False	8	11	39	2015-08-26 14:42:34 IST
15	3655508723	79.42	False	11	11	39	2015-08-26 14:42:23 IST
16	3655508739	252.957	False	1	13	39	2015-08-26 14:42:39 IST
17	3655508733	1.401	False	8	11	39	2015-08-26 14:42:33 IST
18	3655508730	0.0	False	2	11	39	2015-08-26 14:42:30 IST

<< < > >> Go to page: 1 Row count: 25 Showing 1-18 of 18

After performing analytics, you can communicate results in WSO2 DAS as described in the next [Communicating results](#) section below .

Communicating results

The final step in the event flow is to visualize the data. WSO2 DAS uses several presentation mechanisms to present event notifications and processing results. Thereby, it provides the Analytics Dashboard to visualize the processed data for decision making.

The Analytics Dashboard is used to create customizable dashboards for analytics data visualization. Dashboard creation is wizard driven, where you can use gadgets/widgets such as Line, Bar, Arc charts to get data from analytical tables and add them on a structured grid layout to provide an overall view on the analyses. For more information, see [Presenting Data](#) .

Using the Analytics Dashboard

WSO2 DAS provides an [Analytics Dashboard](#) for creating customizable dashboards for visualization of analytical data. Follow the steps below to present data using the Analytics Dashboard.

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.

Creating a Dashboard

You can create a new Dashboard to present the data of the above analytics as shown in the example below. Follow the steps below to create a new Dashboard in the Analytics Dashboard. For more information, see [Adding a Dashboard](#).

1. Click the following **CREATE DASHBOARD** button in the top navigational bar to create a new dashboard.



2. Enter a **Title** and a **Description** for the new dashboard as shown below, and click **Next**.

CREATE A DASHBOARD

Name of your Dashboard

URL

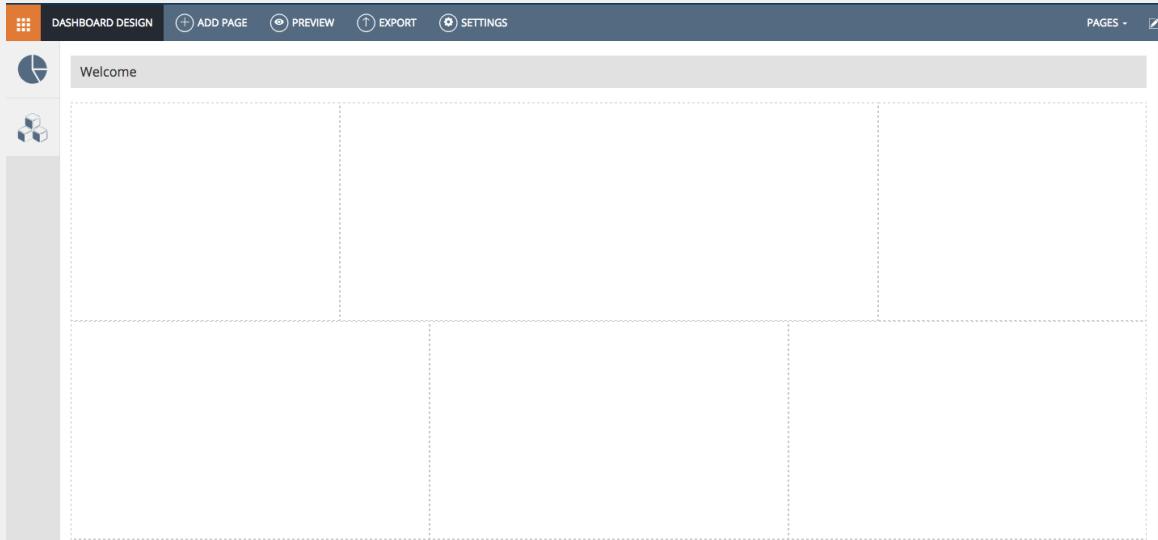
Description

Next

3. Select a layout to place its components as shown below.



4. Click **Select**. You view a layout editor with the chosen layout blocks marked using dashed lines as shown below. Now the dashboard is persisted to the disk.



5. Click the following icon in the top menu.
6. Click **Dashboards** to view the new dashboard added to the list of all available dashboards as shown below.

WSO2 DASHBOARD DESIGNER

DASHBOARDS

* View, Modify and Delete dashboards

Power_Dashboard		
View	Design	Settings

Creating a Gadget

Follow the steps below to create a Bar Chart Gadget to visualize the analyzed data by selecting the event stream created above (i.e. UsageStream) as the data source, and add it to the Power_Dashboard you created above. For more information, see [Adding Gadgets to a Dashboard](#).

1. Log in to the Analytics Dashboard, if you are not already logged in.
2. Click the following **CREATE GADGET** icon in the top menu bar.



- Select the `UsageStream` as the input data source as shown below.

CREATE A GADGET

Select Table/Event Stream

Configure Chart

Analytics Table/Event Stream

USAGESTREAM [batch]

Preview Data

- Select **Chart Type** and enter the preferred x, y axis and additional parameters based on the selected chart type as shown below.

CREATE A GADGET

Select Datasource

Configure Chart

Configuration

Chart Type

Line

Title

Usage_Data

X-Axis

currentTime

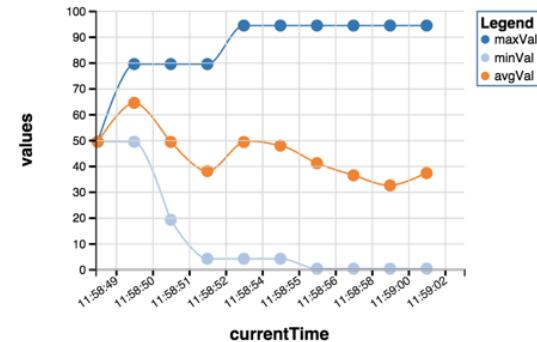
Y-Axis

maxVal
minVal
house_id
avgVal

Interpolation Mode

Monotone

Preview



Preview

Previous

Add to Gadget Store

Next

- Click **Add to Gadget Store** to generate a gadget with the information you provided.
- Click the corresponding **Design** button of the dashboard to which you want to add a gadget as shown below.
- Click the following gadget browser icon in the side menu bar.



You view the new gadget listed in the gadget browser. If not, search for it using its name.

- 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.

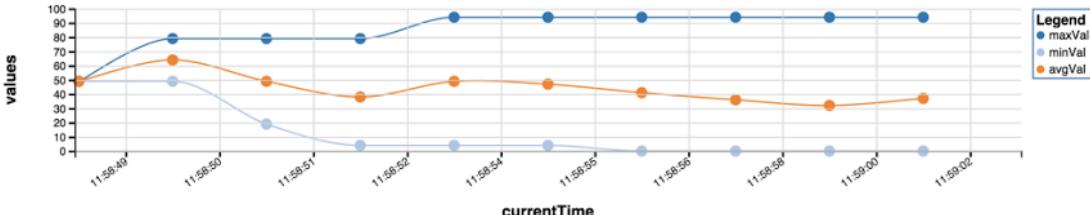
Power_Dashboard

Home

Plug_Usage

admin ▾

Usage_Stats



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"> • ~ 2 GB minimum • ~ 512 MB heap size. This is generally sufficient to process typical SOAP messages, but the requirements vary with larger message sizes and the number of messages processed concurrently.
Disk	<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 or support OpenJDK. • 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 Environments Tested with 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 To launch WSO2 ESB, you need to have Oracle JDK 1.7.*/1.8.*. You cannot launch WSO2 ESB with Oracle JDK 1.6.* or lower.</p>	1.7 or later / 1.8.*	We do not recommend OpenJDK.	http://java.sun.com/javase/downloads/index.jsp
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).	3.0.*	http://maven.apache.org
Web Browser	<p>To access each product's Management Console . The Web Browser must be JavaScript enabled to take full advantage of the Management console.</p> <p>i 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>		

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 a SSH connection to the Linux machine or log in on the text Linux console. You should either log in as root or obtain root permissions after logging in via `su` or `sudo` command.

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.

(i) 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](#).

i 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 Be sure your system meets the .

i Java Development Kit (JDK) is essential to run the product. Also DAS analytics framework depends on Apache Hadoop and Hadoop requires Cygwin, in order to run in Windows.

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

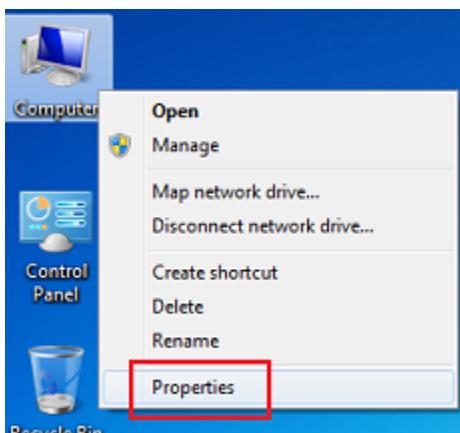
You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer. Typically, the JDK is installed in a directory under `C:/Program Files/Java`, such as `C:/Program Files/Java/jdk1.6.0_27`. If you have multiple versions installed, 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.

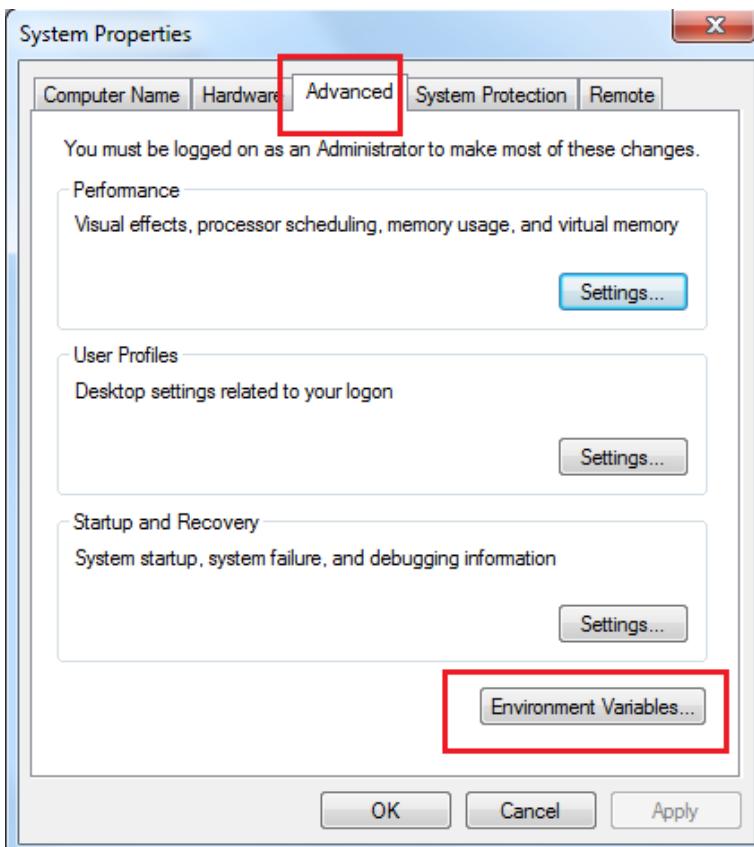
You set up `JAVA_HOME` using the System Properties, as described below. Alternatively, if you just want to set `JAVA_HOME` temporarily for the current command prompt window, [set it at the command prompt](#) .

Setting up JAVA_HOME using the system properties

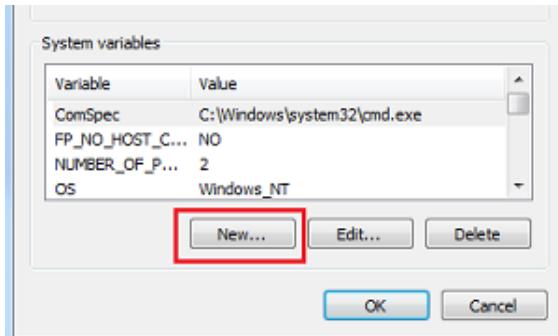
1. Right-click the **My Computer** icon on the desktop and choose **Properties**.



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/jdk1.6.0_27

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/jdk1.6.0_27`

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

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

```
set JAVA_HOME
```

3. The system returns the JDK installation path.

Setting system properties

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

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

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

Installing as a 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.

i 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\ufs_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\ADMINI\LOCALS\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\ADMINI\LOCALS\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

o 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.

i 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

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

Downloading the source

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

After downloading the source of the Carbon Kernel, execute the following command to download the source of the product: `git clone https://github.com/wso2/product-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.

- 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

- Add the required changes to the source code.

Building the product

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

- Make sure the build server has an active Internet connection to download dependencies while building.
- Install Maven and JDK. For compatible versions, see [Installation Prerequisites](#).
- 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 as explained in the [Get Involved](#) section.

Upgrading from WSO2 BAM 2.5.0

This section provides information on how you can upgrade from WSO2 BAM 2.5.0 to WSO2 DAS 3.0.0. For more information on release versions, see the [Release Matrix](#).



WSO2 DAS is not backward compatible with WSO2 BAM

The artifacts of WSO2 DAS are not backward compatible with the artifacts in WSO2 BAM due to the major differences in the architecture between the two products.



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
- Upgrading the database
- Migrating the configurations
- Migrating artifacts

- Testing the upgrade

Preparing to upgrade

The following prerequisites should be completed before upgrading.

- Make a backup of the BAM 2.5.0 database and copy the <BAM_HOME_2.5.0> directory in order to backup the product configurations.
- Download WSO2 DAS 3.0.0 from <http://wso2.com/products/data-analytics-server/>.

Upgrading the database

The instructions in this section describe how you can perform a data migration to upgrade the BAM 2.5.0 database for use in DAS 3.0.0.

1. Before you upgrade to DAS 3.0.0, create a **new** database and restore the backup of the BAM 2.5.0 database in this new database.



Note

You should NOT connect WSO2 DAS to an older database that has not been upgraded.

2. Select the relevant script for the upgrade from [here](#) and run it on the new database. Running this script will ensure that the new database is upgraded to have the additional tables and schemas that are required for WSO2 DAS 3.0.0.



Note

There are three migration scripts available: `migration-service-provider.sql`, `migration-identity.sql` and `migration.sql`. However, only the `migration.sql` script is required to be executed for the DAS database upgrade.

Migrating the configurations

Handling event flow related configurations

Configurable objects used for realtime processing in WSO2 DAS are different to those in WSO2 BAM as explained in the table below. Therefore, they need to be manually reconfigured instead of migrated.

Artifact Type in BAM	Replaced in DAS By	Note	Recommended Action

<ul style="list-style-type: none"> • Input Event Adapter • Event Builder 	Event Receiver	<p>The Event Receiver artifact in DAS 3.0.0 embeds both the Input Event Adapter and Event Builder artifacts in BAM 2.5.0.</p>	<p>Identify the input event adapters and event builders used in combination with the event receiver. For detailed instructions, see Configuring Event Receivers.</p>
<ul style="list-style-type: none"> • Event Stream 	N/A	<p>The Event Stream artifact in DAS is different to that in BAM since a DAS event stream can be a real time event stream or a persistent event stream, whereas BAM event streams are always defined for persistent data.</p>	<p>Redefine the event streams in DAS 3.0.0 using the following procedure.</p> <ol style="list-style-type: none"> 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/StreamDefinition_version> as shown in the example below. 4. Click on the stream version. This will open the Detail View tab for the stream. 5. Click Display as Text to view the event stream configuration as a JSON array. 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. 12. Click Add Event Stream. 13. If you want the event stream to persist events, follow the instructions in Defining Persistent Event Streams. <p>Alternatively, you can redefine the complete event stream configuration in the Available Event Streams page.</p>
<ul style="list-style-type: none"> • Output Event Adapter • Event Formatter 	Event Publisher	<p>The Event Publisher artifact in DAS 3.0.0 embeds both the Output Event Adapter and Event Formatter artifacts in BAM 2.5.0.</p>	<p>Identify the output event adapters and event formatters used in combination with the event publisher. For detailed instructions, see Creating Alerts.</p>

• Execution Plan	N/A	The Siddhi Query Language in DAS 3.0.0 is different to that in BAM 2.5.0. Therefore, the Siddhi queries of an execution plan directly migrated from BAM cannot function in DAS 3.0.0.	Redefine the execution plans in DAS 3.0.0. For detailed instructions on defining execution plans, see Creating Distributed Execution Plan . For the modified Siddhi Query Language, see Siddhi Query Language .
------------------	-----	---	---

Migrating data

Run the `<DAS_HOME>/bin/analytics-migrate.sh` script to migrate data from WSO2 BAM to WSO2 DAS. For detailed information, see [Analytics Migration Tool](#).

 WSO2 BAM uses Cassandra as the event store. In WSO2 DAS, you can select either Cassandra, RDBMS or HBase as the event store. The `analytics-migrate.sh` script migrates Cassandra data to the new event store in WSO2 DAS, however, migrating other databases such as the `User Store` database should be done separately.

Handling indexes

The indexing mechanism in WSO2 DAS is different to that in WSO2 BAM. Therefore, the Cassandra custom and secondary indexes cannot be directly migrated to WSO2 DAS.

Recommended action

WSO2 DAS uses Apache Lucene for indexing. To use the Lucene indexing functionality, select the required stream attributes as index columns as described in [Configuring Indexes](#).

Handling scripts

Scripts that analyze data in WSO2 BAM are written in the Apache Hive query language whereas the scripts in WSO2 DAS are written in the Apache Spark SQL query language. Therefore, the syntax of the scripts differ in WSO2 BAM and WSO2 DAS. Due to this, scripts cannot be directly migrated from WSO2 BAM to WSO2 DAS.

Recommended action

Redefine the required scripts in WSO2 DAS in the Spark query language. For detailed information on writing queries in the Spark SQL query language, see [Spark SQL query language](#).

Handling dashboards and gadgets

The BAM Dashboard in WSO2 BAM is replaced with the Analytics Dashboard in WSO2 DAS. Since the two dashboards and their gadgets are configured differently, dashboards and gadgets cannot be directly migrated from WSO2 BAM to WSO2 DAS.

Recommended action

Redefine the required dashboards and gadgets in the [Analytics Dashboard](#).

Handling reports

The reporting feature in WSO2 BAM is deprecated in WSO2 DAS. Therefore, reports cannot be migrated from BAM to DAS.

Migrating artifacts

BAM toolbox functionality is not available in DAS. Therefore, it is not possible to migrate a toolbox from BAM to DAS. If you need the artifacts in a BAM toolbox for a particular user case, you need to create a corresponding C-App and deploy it in DAS. For more information on creating and deploying C-Apps, see [Packaging Artifacts as a C-App Archive](#).

- The artifacts deployed in the C-App should be DAS artifacts and not BAM artifacts. e.g., The scripts included should be Spark scripts instead of Hive scripts.

Testing the upgrade

1. When the database upgrade scripts are executed, the following are some of the new tables that will be created in the database:
 - UM_DOMAIN
 - UM_SYSTEM_USER
 - UM_SYSTEM_ROLE
 - UM_SYSTEM_USER_ROLE
2. Deploy the C-Apps you created to correspond with the relevant BAM toolboxes and check whether all the required artifacts are successfully created in your DAS environment.
3. [Publish data to WSO2 DAS](#) and check whether they are received by the [event receivers](#) configured in DAS.
4. Execute the scripts you redefined in the Spark SQL query language, and use the [Data Explorer](#) and/or the [Analytics Dashboard](#) to view the results.
5. 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

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

1. Open a command prompt by following the instructions below.

- On Windows: Click **Start -> Run**, type cmd at the prompt, and then press **Enter**.
- On Linux/Solaris/Mac OS: Establish an SSH connection to the server, log in to the text Linux console, or open a terminal window.

2. Execute one of the following commands:

- To start the server in a typical environment:
 - On Windows: <PRODUCT_HOME>\bin\wso2server.bat --run
 - On Linux/Solaris/Mac OS: sh <PRODUCT_HOME>/bin/wso2server.sh
- To start the server in the background mode of Linux: sh <PRODUCT_HOME>/bin/wso2server.sh start
To stop the server running in this mode, you will enter: sh <PRODUCT_HOME>/bin/wso2server.sh stop
- To provide access to the production environment without allowing any user group (including admin) to log into the Management Console:
 - On Windows: <PRODUCT_HOME>\bin\wso2server.bat --run -DworkerNode
 - On Linux/Solaris/Mac OS: sh <PRODUCT_HOME>/bin/wso2server.sh -DworkerNode
- To check for additional options you can use with the startup commands, type -help after the command, such as:
sh <PRODUCT_HOME>/bin/wso2server.sh -help (see the related topics section at the end of this page).

3. 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".

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.

- i 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>
```

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>
```

i Any configurations (including valves 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
```

Get Involved

WSO2 products are 100% open source and released under the Apache License Version 2.0. WSO2 welcomes anyone who is interested in WSO2 products to become a contributor by getting involved in the WSO2 community and helping with the development of WSO2 projects.

How can I get involved in the community?

Contributing as a non-committer – anyone can do it!

- Overview of the WSO2 repository
- Contributing to the WSO2 code base
- [WSO2 GitHub Guidelines](#)

How can I get involved in the community?

You can get involved in the WSO2 community in various ways:

- **Use WSO2 products**

The latest binary packs that correspond to the WSO2 product releases can be downloaded freely via the respective product pages on the [WSO2 website](#). We recommend that you download and use WSO2 products so that you can discover the advantages of our lean middleware stack. Your feedback on our products is much appreciated, as it will help us to drive our product roadmaps and the underlying technology. For information on product releases, go to the [Release Matrix](#). For tutorials, articles, white papers, webinars, WSO2 documentation, and other learning resources, look in the Resources menu on the [WSO2 website](#).

- **Join WSO2 mailing lists**

Many WSO2 mailing lists are open to the public, so anyone interested in WSO2 products can monitor the mail threads. You can subscribe to the dev@wso2.org and architecture@wso2.org mailing lists to get involved in the discussions on WSO2 development. For more information on subscribing to these mailing lists, see [WSO2 Mailing Lists](#).

- **Participate in user forums**

The WSO2 team monitors and participates in the discussions on [Stack Overflow](#). If you have any technical or programming questions related to WSO2 products, post them on Stack Overflow. Be sure to tag your question with appropriate keywords such as WSO2 and the product name so that our team can easily find your questions and provide answers. If you cannot find an answer on the user forum, you can email the WSO2 development team directly using the relevant mailing lists described at [WSO2 Mailing Lists](#). We also encourage you to contribute by answering your fellow users' questions on Stack Overflow.

- **Report bugs**

WSO2 has a public bug-tracking system that you can use to report issues, submit enhancement requests, and track and comment on issues. You can also use this system to report issues related to [WSO2 product documentation](#). If you find a bug, first search the dev mailing list to see if someone has faced the same issue, and also check the bug-tracking system. If you can't find any information on the issue, create an issue in the bug-tracking system with as much information as possible, including the product version you were using, how to reproduce the issue, etc.

- **Contribute to the WSO2 code base**

WSO2 invites you to contribute by providing patches for bug fixes or features. For this purpose you can check out the source of the relevant GitHub repository, build the product, and make changes. You can then contribute your changes by sending a [pull a request](#) for review. For more information, see the next section.

Contributing as a non-committer – anyone can do it!

Anyone (not just committers) can share contributions to WSO2's open-source software products. Your work will be recognized: if your contribution – feature enhancement, bug fix, or other improvements – is accepted, your name will be included as an author in the official commit logs. Read on for details on how you can contribute.

Overview of the WSO2 repository

WSO2 uses [Git](#) as its source control management system. The [WSO2 Git repository](#) maintains the code repository and the active build for continuous delivery incorporated with integrated automation.

Contributing to the WSO2 code base

Follow these instructions to contribute to the WSO2 code base. Be sure to follow the [WSO2 GitHub Guidelines](#).

1. [Fork](#) the respective code base to your Git account.
2. Clone the code base to your local machine.

```
git clone <GitHub-REPOSITORY-URL>
```

If you are not sure which repository needs to be cloned, send an email to dev@wso2.org.

3. Build the product using Maven.

Prerequisites

- Install Maven and JDK. See the [Installation Prerequisites](#) page for compatible versions.

- Set the environment variable MAVEN_OPTS="-Xms768m -Xmx3072m -XX:MaxPermSize=1200m" to avoid the maven OutOfMemoryError.
- Make sure the build server has an active Internet connection to download dependencies while building.

Use the following commands to create complete release artifacts of a WSO2 product, including the binary and source distributions.

Command	Description
mvn clean install	The binary and source distributions.
mvn clean install -Dmaven.test.skip=true	The binary and source distributions, without running any of the unit tests.
mvn clean install -Dmaven.test.skip=true -o	The binary and source distributions, without running any of the unit tests, in offline mode. This can be done only if you've already built the source at least once.

4. If you need to add a new file to the repository:

- Add the new file.

```
git add <FILE-NAME>
```

For example:

```
git add mycode.java
```

- Commit the newly added file to your local repository.

```
git commit -m "<COMMIT-MESSAGE>"
```

For example:

```
git commit -m "Adding a new file"
```

5. If you need to update an existing file in the repository:

- Open the file that you want to update and make the necessary changes.

- Commit the changes to your local repository.

```
git commit -m "<COMMIT-MESSAGE>" -a
```

For example:

```
git commit -m "Updated the clauses in the terms and conditions file" -a
```

6. Sync your changes with the upstream repository.

```
git remote add <TAG-NAME> <UPSTREAM-GIT-REPO-URL>
git fetch <TAG-NAME>
git merge <TAG-NAME>/<BRANCH-NAME>
```

For example:

```
git remote add wso2_upstream https://github.com/wso2/wso2-synapse.git
git fetch wso2_upstream
git merge wso2_upstream/master
```

7. Push the changes to your own Git repository.

```
git push
```

8. Send a [Git pull request](#) to the WSO2 Git repository and add the URL of the Git pull request in the JIRA that corresponds to the patch. Your pull request will be authorized only after it is reviewed by the team lead or release manager or responsible person for the corresponding Git repository.

For more information on using GitHub, see the related help articles [Fork A Repo](#) and [Using Pull Requests](#).

WSO2 GitHub Guidelines

- **The respective WSO2 Git repository should be forked**

When contributing to WSO2 code base by way of a patch, make sure you identify the correct Git repository that needs to be forked. For more information on WSO2 Git repositories, see [WSO2 GitHub Repositories](#). If you still are not sure which repository needs to be cloned, send an email to dev@wso2.org so that a WSO2 team member can advise you.

- **Do not build any dependencies**

You do not need to build any dependencies, as everything you need will be automatically fetched from the Maven repository (Nexus) when you are building the product on your machine. Make sure the build server has an active Internet connection to download dependencies while building.

- **Always sync with the forked repository before issuing a pull request**

There is a high possibility that the forked repository may differ from the upstream repository (remote repository that was forked) that you initially forked. Therefore, always sync the repository to prevent pull requests from being rejected.

WSO2 GitHub Repositories

The following are the WSO2 GitHub repositories that need to be forked, so that you can contribute to the WSO2 community by offering patches for bug fixes or features for WSO2 products.

- Kernel level Git repositories
- Platform level Git repositories
- Mobile platform Git repositories
- Product level Git repositories
- Other WSO2 Git repositories

Kernel level Git repositories

Repo URL	Description
carbon-kernel	Carbon 5 kernel repo
carbon4-kernel	Carbon 4 kernel repo

Platform level Git repositories

Repo URL	Description
carbon-analytics	Contains components and features related to analytics services.
carbon-apimgt	Contains components and features related to API management.
carbon-appmgt	Contains components and features related to application management.
carbon-business-messaging	Contains the components and features related to business messaging.
carbon-business-process	Contains components and features related to business processes.
carbon-commons	Contains common components and features shared across the platform projects.
carbon-data	Contains components and features related to data services.
carbon-deployment	Contains components and features related to web application and service development (i.e., JavaEE WebProfile support, JAX-WS/RS service deployment, Webapp monitoring dashboards etc.).
carbon-event-processing	Contains components and features related to event processing services.
carbon-governance	Contains components and features related to governance services.
carbon-mediation	Contains components and features related to mediation services.

carbon-ml	Contains components and features related to machine learner.
carbon-multitenancy	
carbon-parent	
carbon-platform-automated-test-suite	Contains WSO2 product integration test suites and Platform test suites with ant based test executor.
carbon-platform-integration	Contains WSO2 test automation framework modules.
carbon-platform-integration-utils	Contains utilities related to WSO2 test automation framework which is common to the whole product platform.
carbon-qos	Contains components and features related to quality of service.
carbon-registry	Contains components and features related to registry services.
carbon-rules	Contains components and features related to business rules.
carbon-storage-management	Contains sources corresponding to the components that are primarily being used for storage provisioning and management related tasks. Out of all the components being maintained within this particular repository some components (i.e., Cassandra, HDFS) are used across the platform. In addition, some of the tools developed for storage browsing (i.e., Cassandra-Explorer etc.) too are part of this repository.
carbon-store	
carbon-utils	Contains ntask, remote-tasks, ndatasource etc.

Mobile platform Git repositories

Repo URL	Description
emm-agent-android	Maintains the Android agent that is used to enroll the device to EMM server.
emm-agent-ios	Maintains the iOS agent that is used to enroll the device to EMM server.

Product level Git repositories

Product Name	Repo URL	Description
API Manager	product-apim	Maintains sources corresponding to building and packaging of WSO2 API manager distribution.
App Factory	product-af	Maintains sources corresponding to building and packaging of WSO2 APP Factory distribution.
Application Server	product-as	Maintains sources corresponding to building and packaging of WSO2 Application Server distribution.
Business Activity Monitor	product-bam	Maintains sources corresponding to building and packaging of WSO2 Business Activity Monitor distribution.
Business Process Server	product-bps	Maintains sources corresponding to building and packaging of WSO2 Business Process Server distribution.
Business Rules Server	product-brs	Maintains sources corresponding to building and packaging of WSO2 Business Rules Server distribution.

Complex Event Processor	product-cep	Maintains sources corresponding to building and packaging of WSO2 Complex Event Processor distribution.
Connected Device Management Framework	product-cdm	Maintains sources corresponding to building and packaging of WSO2 Connected Device Management Framework distribution .
Data Analytics Server	product-bam	Maintains sources corresponding to building and packaging of WSO2 Data Analytics Server distribution.
Data Services Server	product-dss	Maintains sources corresponding to building and packaging of WSO2 Data Services Server distribution.
Enterprise Mobility Manager	product-emm	Maintains sources corresponding to building and packaging of WSO2 Enterprise Mobility Manager distribution.
Enterprise Service Bus	product-esb	Maintains sources corresponding to building and packaging of WSO2 Enterprise Service Bus distribution.
Enterprise Store	product-es	Maintains sources corresponding to building and packaging of WSO2 Enterprise Store distribution.
Governance Registry	product-greg	Maintains sources corresponding to building and packaging of WSO2 Governance Registry distribution.
Identity Server	product-identity	Maintains sources corresponding to building and packaging of WSO2 Identity Server distribution.
Message Broker	product-mb	Maintains sources corresponding to building and packaging of WSO2 Message Broker distribution.
Private PaaS	product-private-paas	Maintains sources corresponding to building and packaging of WSO2 Private PaaS distribution.
Storage Server	product-ss	Maintains sources corresponding to building and packaging of WSO2 Storage Server distribution.
Task Server	product-ts	Maintains sources corresponding to building and packaging of WSO2 Task Server distribution.
Developer Studio	developer-studio	Maintains sources corresponding to building and packaging of WSO2 Developer Studio distribution.

Other WSO2 Git repositories

The following are GitHub repository URLs that correspond to independent projects managed by WSO2:

Repo URL	Description
andes	Message broker core engine implementation.
balana	XACML core engine implementation.
charon	SCIM core engine implementation.
esb-connectors	Collection of connectors that allows you to interact with WSO2 ESB's third-party product function.
jaggery	This repo contains Jaggeryjs. Jaggery is a framework used to write webapps and HTTP-focused web services for all aspects of the application: front-end, communication, Server-side logic and persistence in pure Javascript.
jaggery-extensions	This contains extensions for the Jaggery framework.

orbit	Used to create OSGi bungles out of third-part dependencies.
siddhi	Complex event processing core engine implementation.

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

[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>The data the event contains. Data is divided into the following three categories for maintenance and usability. (Define at least one event steam attribute.)</p> <ul style="list-style-type: none"> Meta Data: Contains the meta information of the events. (Referred to as <code>meta_<attribute name></code>.) Correlation Data: Contains the correlation information of the events. (Referred to as <code>correlation_<attribute name></code>.) Payload Data: Contains the actual data that the event intends to have. (Referred to as <code><attribute name></code>.)

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.

Attribute Name	Attribute Type	Actions
ip	string	Delete

Attribute Name	Attribute Type	Actions
id	long	Delete

Attribute Name	Attribute Type	Actions
testMessage	string	Delete

4. Click **Add Event Stream**, to create the Event Stream in the system. When you click **OK** in the pop-up message on successful addition of the stream definition, you view it in the **Available Event Streams** list as shown below.

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.

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



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

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

[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](#)

5. 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.

1. Log in to the management console, and click **Main**.

2. Click **Event Streams** in the **Event Processor** menu. You view the **Available Event Streams** list.
3. Click the **Delete** button of the corresponding event stream to delete it.

Editing an event stream

Follow the steps below to edit an event stream by editing the corresponding event stream definition.

1. Log in to the management console, and click **Main**.
2. Click **Event Streams** in the **Event Processor** menu. You view the **Available Event Streams** list.
3. Click the **Edit** button of the corresponding event stream to edit it.

 Click the **switch to source view** link to edit an event stream using the source view.

Creating sample events

Follow the steps below to create sample events for a defined event stream.

1. Log in to the management console, and click **Main**.
2. Click **Event Streams** in the **Event Processor** menu. You view the **Available Event Streams** list.
3. Click the **Event Stream Id** of the corresponding event stream for which you want to create the sample event.
4. Select the event format type (i.e. **xml**, **json**, or **text**) from the drop down list, which you want to create the sample event in.
5. You view details of the event stream as shown below.



```
{
  "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

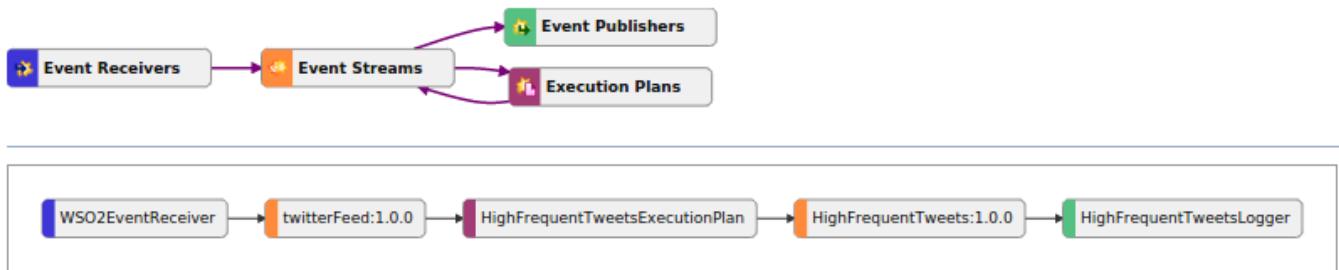
Event flow visualizes the stream flow in WSO2 CEP/DAS to easily navigate to different WSO2 CEP/DAS components.

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	false

merge.schema	<p>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.</p> <p>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</p>	No	true
--------------	--	----	------

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). The DAL (Analytics Data Service) is made up of two main components which are specified in the <DAS_HOME>/repository/conf/analytics/analytics-config.xml file as follows.

- Analytics Record Store
- Analytics File System

sample analytics-config.xml

```
<analytics-dataservice-configuration>
    <!-- The name of the primary record store -->
    <primaryRecordStore>EVENT_STORE</primaryRecordStore>
    <!-- The name of the index staging record store -->
    <indexStagingRecordStore>INDEX_STAGING_STORE</indexStagingRecordStore>
    <!-- Analytics File System - properties related to index storage implementation -->
    <analytics-file-system>

        <implementation>org.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsFileSystem</implementation>
            <properties>
                <!-- the data source name mentioned in data sources configuration -->
                <property name="datasource">WSO2_ANALYTICS_FS_DB</property>
                <property name="category">large_dataset_optimized</property>
            </properties>
        </analytics-file-system>
        <!-- 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_STORE_DB</property>
                    <property name="category">large_dataset_optimized</property>
                </properties>
            </analytics-record-store>
            <analytics-record-store name="INDEX_STAGING_STORE">

                <implementation>org.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsRecordStore</implementation>
```

```

implementation>
    <properties>
        <property name="datasource">WSO2_ANALYTICS_EVENT_STORE_DB</property>
        <property name="category">limited_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 maximum number of threads used for indexing per node, -1 signals to auto
detect the optimum value,
        where it would be equal to (number of CPU cores in the system - 1) -->
    <indexingThreadCount>-1</indexingThreadCount>
    <!-- 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 number of batch index records, the indexing node will process per each
indexing thread. A batch index record basically
        encapsulates a batch of records retrieved from the receiver to be indexed -->
    <shardIndexRecordBatchSize>100</shardIndexRecordBatchSize>
    <!-- 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 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>
<!-- Receiver/Indexing flow-control configuration -->
<analytics-receiver-indexing-flow-control enabled="true">
    <!-- maximum number of records that can be in index staging area before
receiving is throttled -->
    <recordReceivingHighThreshold>10000</recordReceivingHighThreshold>
    <!-- the limit on number of records to be lower than, to reduce throttling -->
    <recordReceivingLowThreshold>5000</recordReceivingLowThreshold>

```

```
</analytics-receiver-indexing-flow-control>
</analytics-dataservice-configuration>
```

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.
Index Staging Store	INDEX_STAGING_STORE	This record store is used to store meta data that need to be saved before writing indexed data into the file system.
Processed Data Store.	PROCESSED_DATA_STORE	This record store is used to store summarised 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. For the record store to function, the provider for the datasource type mentioned in this implementation should be enabled in the <DAS_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	Default Value	
EVENT_STORE	INDEX_STAGING_STORE	PROCESSED_DATA_STORE	
datasource	The name of the datasource used to connect to the database used by the record store.	WSO2_ANALYTICS_EVENT_STORE_DB	WSO2_ANAI

category	Possible values are as follows: <ul style="list-style-type: none"> • large_dataset_optimized : If this property value is added the record store is more suitable to be used by event streams with a high load of events. • limited_dataset_optimized: If this property value is added, the record store is more suitable to be used by event streams which handle relatively few events. 	large_dataset_optimized	limited_dataset_optimized
----------	--	-------------------------	---------------------------

i 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 File System

Analytics File System represents a storage area used for storing index data. The index data is generated from the records added to DAS. A set of background tasks are continuously run to look up data in Analytics Record Store and identify the tables that should be indexed. These tables are identified based on the information entered when [persisting event streams](#). Then the index processing is carried out using Apache Lucene, and the resulting data is stored in the Analytics File System.

Configuring the file system

The following is a sample configuration of a File System in the `analytics-config.xml` file.

```
<analytics-file-system>

<implementation>org.wso2.carbon.analytics.datasource.rdbms.RDBMSAnalyticsFileSystem</implementation>
<properties>
    <!-- the data source name mentioned in data sources configuration -->
    <property name="datasource">WSO2_ANALYTICS_FS_DB</property>
    <property name="category">large_dataset_optimized</property>
</properties>
</analytics-file-system>
```

The following needs to be specified in a file system configuration.

- **Implementation:** This specifies the implementation for the file system. For the file system to function, the provider for the datasource type mentioned in this implementation should be enabled in the `<Das_Home>/repository/conf/datasources/analytics-datasources.xml` file.
- **File system specific properties:** The properties that are defined for a file system are described in the table below.

Property	Description	Default Value
EVENT_STORE	INDEX_STAGING_STORE	PROCESSED_DATA_STORE

datasource	The name of the datasource used to connect to the database used by the file system.	WSO2_ANALYTICS_EVENT_STORE_DB	WSO2_AN
category	<p>Possible values are as follows:</p> <ul style="list-style-type: none"> • <code>large_dataset_optimized</code>: If this property value is added the file system is more suitable to be used by event streams with a high load of events. • <code>limited_dataset_optimized</code> : If this property value is added, the file system is more suitable to be used by event streams which handle relatively few events. 	large_dataset_optimized	limited

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

The following parameters are common for both the Analytics Record Store and the Analytics File System

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.	
<code><data-retention-days></code>	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

Flow control parameters

Parameter	Description	Default Value
<code><recordReceivingHighThreshold></code>	The minimum number of records that should be accumulated in the index staging record store in order to stop further records from being written into it.	10000

<recordReceivingLowThreshold>	The maximum number of records that should be accumulated in the index staging record store in order to allow further records to be written into it. This is relevant in a situation where receiving further records is throttled as a result of the number of records reaching the value specified for the <code>recordReceivingHighThreshold</code> parameter.	5000
-------------------------------	---	------

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</implementation>
<indexingThreadCount>	The maximum number of threads used for indexing per node. When -1 is optimum value (equal to the number of cores in the system) is automatically the number of threads is generated accordingly.
<shardCount>	The number of index shards the server should maintain per cluster. This scaling nature of the indexing cluster. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> ⚠ This parameter can only be set once for the lifetime of the cluster, and cannot be changed later on. </div>
<shardIndexRecordBatchSize>	The number of batch index records the indexing node should process per thread at a given time. <p>An index record contains data of a record batch inserted in a single put batch can be as high as the event receiver queue data size, which is 100MB. Therefore, the highest amount of in-memory record data that an indexing process can have is 10MB * 100. This parameter should be configured to change the amount of memory available to the indexing node based on your requirements.</p> <p>The above implementations can be done by general users and can be plug-in to the server. And allows implementors to provide new, as well as standard implementations on top of existing implementations to provide additional enhancements such as data encryption, custom auditing etc..</p> <p>The above two interfaces can be found in <code>DAS_HOME/repository/components/org.wso2.carbon.analytics.datasource.core-* .jar</code>.</p>

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/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 (version 2.0.0 and upwards) is required. The

Cassandra implementation for the DAS DAL contains both the implementations of Analytics Record Store and Analytics File System.

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

- Enabling the Cassandra datasource provider
- Configurations for the Analytics Record Store
- Configurations for the Analytics File System
- Configuring the datasources

 The Cassandra datasource does not support pagination or record count.

Enabling the Cassandra datasource provider

In order to use Cassandra as the record store/file system, 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 org.wso2.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.

Configurations for the Analytics File System

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

```
<analytics-file-system>
    <implementation>org.wso2.carbon.analytics.datasource.cassandra.CassandraAnalyticsFileS
ystem</implementation>
    <properties>
        <!-- the data source name mentioned in data sources configuration -->
        <property name="datasource">WSO2_ANALYTICS_DS_CASSANDRA</property>
    </properties>
</analytics-file-system>
```

The properties of the above configuration are described below.

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

Configuring the datasources

Datasources to connect to the above underlying Cassandra implementations should be defined as described below. Configuring the datasource for the Analytics Record Store

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 the datasource for the Analytics File System datasource

Change the configurations of the WSO2_ANALYTICS_FS_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/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 1.0.0 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**), and HDFS (the distributed file system used by Apache Hadoop) for storing index information (**Analytics File System**). 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
- Configurations for the Analytics File System
- Configuring the datasources



The HBase datasource does not support pagination or record count.

⚠ It is required that all HBase/HDFS nodes and all DAS nodes are time synced. Utilities such as `ntp` 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</provider>-->
```

2. Uncomment the HBase datasource provider as shown below.

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

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

Configurations for the Analytics File System

For configuring HBase as the underlying datasource implementation for the Analytics File System, specify the HDFS configurations in the `<DAS_HOME>/repository/conf/analytics/analytics-config.xml` file as shown in

the below example.

```
<analytics-file-system>

<implementation>org.wso2.carbon.analytics.datasource.hbase.HDFSAnalyticsFileSystem</implementation>
<properties>
    <!-- the data source name mentioned in data sources configuration -->
    <property name="datasource">WSO2_ANALYTICS_FS_DB_HDFS</property>
</properties>
</analytics-file-system>
```

The properties of the above configurations are described below.

Property	Description
<implementation>	The implementation class of the Analytics File System relevant for HDFS, which is <code>org.wso2.carbon.analytics.datasource.hbase.HDFSAnalyticsFileSystem</code> .
<property name="datasource">	The Carbon datasource name of the type "HDFS", which is used to look up to find the associated HDFS data source.

Configuring the datasources

You need to define datasources to connect to the above underlying HBase and HDFS implementations as described below.

Configuring the datasource for the Analytics Record Store

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 HBase Datasource](#).

Configuring the datasource for the Analytics File System datasource

Change the configurations of the `WSO2_ANALYTICS_FS_DB` datasource in the `<DAS_HOME>/repository/conf/datasources/analytics-datasources.xml` file accordingly. For information on the datasource configurations, see [Configuring a HDFS 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 both the implementations of Analytics Record Store and Analytics File System. The following sections describe both implementations of the Cassandra DAL component

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

i 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-configuration.xml` file to `true`.

Enabling the RDBMS datasource provider

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

i 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-config.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 org.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 org.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.

Configurations for the Analytics File System

To configure RDBMS as the underlying datasource implementation for the Analytics Record Store, specify the RDBMS 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.rdbms.RDBMSAnalyticsFileSystem</i
mplementation>
<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 configurations are described below.

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

RDBMS query configuration

Both the above RDBMS Analytics File System and 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 <DA_S_HOME>/repository/conf/analytics/rdbms-query-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 - limited_dataset_optimized categoryoracle

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>
    <fsDataChunkSize>10240</fsDataChunkSize>
    <fsDeletePathQuery>DELETE FROM AN_FS_PATH WHERE path = ?</fsDeletePathQuery>
    <fsDeleteDataQuery>DELETE FROM AN_FS_DATA WHERE path = ?</fsDeleteDataQuery>
    <fsUpdateDataPathQuery>UPDATE AN_FS_DATA SET path = ? WHERE path =
?</fsUpdateDataPathQuery>
    <fsFileLengthRetrievalQuery>SELECT length FROM AN_FS_PATH WHERE path =
?</fsFileLengthRetrievalQuery>
    <fsInsertPathQuery>INSERT INTO AN_FS_PATH
(path,is_directory,length,parent_path) VALUES (?, ?, ?, ?)</fsInsertPathQuery>
    <fsListFilesQuery>SELECT path FROM AN_FS_PATH WHERE parent_path =
?</fsListFilesQuery>
    <fsPathRetrievalQuery>SELECT * FROM AN_FS_PATH WHERE path =
?</fsPathRetrievalQuery>
    <fsReadDataChunkQuery>SELECT data FROM AN_FS_DATA WHERE path = ? AND sequence
= ?</fsReadDataChunkQuery>
    <fsSetFileLengthQuery>UPDATE AN_FS_PATH SET length = ? WHERE path =
?</fsSetFileLengthQuery>
    <fsTableInitQueries>
        <query>CREATE TABLE AN_FS_PATH (path VARCHAR(256), is_directory BOOLEAN,

```

```
length BIGINT, parent_path VARCHAR(256), PRIMARY KEY(path), FOREIGN KEY (parent_path)
REFERENCES AN_FS_PATH(path))</query>
<query>CREATE TABLE AN_FS_DATA (path VARCHAR(256), sequence BIGINT, data
BINARY, PRIMARY KEY (path,sequence), FOREIGN KEY (path) REFERENCES
AN_FS_PATH(path))</query>
<query>CREATE INDEX index_parent_id ON AN_FS_PATH(parent_path)</query>
</fsTableInitQueries>
<fsTablesCheckQuery>SELECT path FROM AN_FS_PATH WHERE path =
'/'</fsTablesCheckQuery>
```

```

<fsMergeDataChunkQuery>MERGE INTO AN_FS_DATA (path,sequence,data) KEY
(path,sequence) VALUES (?,?,?)</fsMergeDataChunkQuery>
</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>
    <fsDataChunkSize>10240</fsDataChunkSize>
    <fsDeletePathQuery>DELETE FROM AN_FS_PATH WHERE path = ?</fsDeletePathQuery>
    <fsDeleteDataQuery>DELETE FROM AN_FS_DATA WHERE path = ?</fsDeleteDataQuery>
    <fsUpdateDataPathQuery>UPDATE AN_FS_DATA SET path = ? WHERE path =
?</fsUpdateDataPathQuery>
    <fsFileLengthRetrievalQuery>SELECT length FROM AN_FS_PATH WHERE path =
?</fsFileLengthRetrievalQuery>
    <fsInsertPathQuery>INSERT INTO AN_FS_PATH
(path,is_directory,length,parent_path) VALUES (?,?,?,?,?)</fsInsertPathQuery>
    <fsListFilesQuery>SELECT path FROM AN_FS_PATH WHERE parent_path =
?</fsListFilesQuery>
    <fsPathRetrievalQuery>SELECT * FROM AN_FS_PATH WHERE path =
?</fsPathRetrievalQuery>
    <fsReadDataChunkQuery>SELECT data FROM AN_FS_DATA WHERE path = ? AND sequence
= ?</fsReadDataChunkQuery>

```

```
<fsSetFileLengthQuery>UPDATE AN_FS_PATH SET length = ? WHERE path =
?</fsSetFileLengthQuery>
<fsTableInitQueries>
    <query>CREATE TABLE AN_FS_PATH (path VARCHAR(256), is_directory BOOLEAN,
length BIGINT, parent_path VARCHAR(256), PRIMARY KEY(path)) ENGINE='MyISAM'</query>
    <query>CREATE TABLE AN_FS_DATA (path VARCHAR(256), sequence BIGINT, data
LONGBLOB, PRIMARY KEY (path,sequence)) ENGINE='MyISAM'</query>
    <query>CREATE INDEX index_parent_id ON AN_FS_PATH(parent_path)</query>
</fsTableInitQueries>
<fsTablesCheckQuery>SELECT path FROM AN_FS_PATH WHERE path =
'/'</fsTablesCheckQuery>
```

```

<fsMergeDataChunkQuery>INSERT INTO AN_FS_DATA (path,sequence,data) VALUES (?, ?, ?) ON DUPLICATE KEY UPDATE data=VALUES(data)</fsMergeDataChunkQuery>
</database>

```

```

<database name = "mysql" category = "limited_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='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>
    <fsDataChunkSize>10240</fsDataChunkSize>
    <fsDeletePathQuery>DELETE FROM AN_FS_PATH WHERE path = ?</fsDeletePathQuery>
    <fsDeleteDataQuery>DELETE FROM AN_FS_DATA WHERE path = ?</fsDeleteDataQuery>
    <fsUpdateDataPathQuery>UPDATE AN_FS_DATA SET path = ? WHERE path = ?</fsUpdateDataPathQuery>
    <fsFileLengthRetrievalQuery>SELECT length FROM AN_FS_PATH WHERE path = ?</fsFileLengthRetrievalQuery>
    <fsInsertPathQuery>INSERT INTO AN_FS_PATH (path,is_directory,length,parent_path) VALUES (?, ?, ?, ?)</fsInsertPathQuery>
    <fsListFilesQuery>SELECT path FROM AN_FS_PATH WHERE parent_path = ?</fsListFilesQuery>
    <fsPathRetrievalQuery>SELECT * FROM AN_FS_PATH WHERE path = ?</fsPathRetrievalQuery>
    <fsReadDataChunkQuery>SELECT data FROM AN_FS_DATA WHERE path = ? AND sequence = ?</fsReadDataChunkQuery>

```

```
<fsSetFileLengthQuery>UPDATE AN_FS_PATH SET length = ? WHERE path =
?</fsSetFileLengthQuery>
<fsTableInitQueries>
    <query>CREATE TABLE AN_FS_PATH (path VARCHAR(256), is_directory BOOLEAN,
length BIGINT, parent_path VARCHAR(256), PRIMARY KEY(path)) ENGINE='InnoDB'</query>
    <query>CREATE TABLE AN_FS_DATA (path VARCHAR(256), sequence BIGINT, data
LONGBLOB, PRIMARY KEY (path,sequence)) ENGINE='InnoDB'</query>
    <query>CREATE INDEX index_parent_id ON AN_FS_PATH(parent_path)</query>
</fsTableInitQueries>
<fsTablesCheckQuery>SELECT path FROM AN_FS_PATH WHERE path =
'/'</fsTablesCheckQuery>
```

```

<fsMergeDataChunkQuery>INSERT INTO AN_FS_DATA (path,sequence,data) VALUES (?, ?, ?) ON DUPLICATE KEY UPDATE data=VALUES(data)</fsMergeDataChunkQuery>
</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 rownum
RNUM, record_id, timestamp, data FROM {{TABLE_NAME}} WHERE partition_key >= ? and
partition_key < ? AND timestamp >= ? AND timestamp < ? and rownum <= ?)
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>
    <fsDataChunkSize>10240</fsDataChunkSize>
    <fsDeletePathQuery>DELETE FROM AN_FS_PATH WHERE path = ?</fsDeletePathQuery>
    <fsDeleteDataQuery>DELETE FROM AN_FS_DATA WHERE path = ?</fsDeleteDataQuery>
    <fsUpdateDataPathQuery>UPDATE AN_FS_DATA SET path = ? WHERE path =
?</fsUpdateDataPathQuery>
    <fsFileLengthRetrievalQuery>SELECT length FROM AN_FS_PATH WHERE path =
?</fsFileLengthRetrievalQuery>
    <fsInsertPathQuery>INSERT INTO AN_FS_PATH

```

```
(path,is_directory,length,parent_path) VALUES (?,?,?,?,?)</fsInsertPathQuery>
    <fsListFilesQuery>SELECT path FROM AN_FS_PATH WHERE parent_path =
?</fsListFilesQuery>
    <fsPathRetrievalQuery>SELECT * FROM AN_FS_PATH WHERE path =
?</fsPathRetrievalQuery>
    <fsReadDataChunkQuery>SELECT data FROM AN_FS_DATA WHERE path = ? AND sequence
= ?</fsReadDataChunkQuery>
    <fsSetFileLengthQuery>UPDATE AN_FS_PATH SET length = ? WHERE path =
?</fsSetFileLengthQuery>
    <fsTableInitQueries>
        <query>CREATE TABLE AN_FS_PATH (path VARCHAR2(256), is_directory
NUMBER(1), length NUMBER(19), parent_path VARCHAR2(256), PRIMARY KEY(path), FOREIGN
KEY (parent_path) REFERENCES AN_FS_PATH(path))</query>
        <query>CREATE TABLE AN_FS_DATA (path VARCHAR2(256), sequence NUMBER(19),
data BLOB, PRIMARY KEY (path,sequence), FOREIGN KEY (path) REFERENCES
AN_FS_PATH(path))</query>
        <query>CREATE INDEX index_parent_id ON AN_FS_PATH(parent_path)</query>
    </fsTableInitQueries>
    <fsTablesCheckQuery>SELECT path FROM AN_FS_PATH WHERE path =
'/'</fsTablesCheckQuery>
    <fsUpdateDataChunkQuery>UPDATE AN_FS_DATA SET data = ? WHERE path = ? AND
sequence = ?</fsUpdateDataChunkQuery>
    <fsWriteDataChunkQuery>INSERT INTO AN_FS_DATA (path,sequence,data) VALUES
(?,?,?,?)</fsWriteDataChunkQuery>
    <fsMergeDataChunkQuery>MERGE INTO AN_FS_DATA dest USING( SELECT ? path, ?
sequence, ? data FROM dual) src ON(dest.path = src.path AND dest.sequence =
src.sequence) WHEN NOT MATCHED THEN INSERT(path, sequence, data) VALUES(src.path,
```

```

src.sequence, src.data) WHEN MATCHED THEN UPDATE SET dest.data =
src.data</fsMergeDataChunkQuery>
</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>
    <fsDataChunkSize>10240</fsDataChunkSize>
    <fsDeletePathQuery>DELETE FROM AN_FS_PATH WHERE path = ?</fsDeletePathQuery>
    <fsDeleteDataQuery>DELETE FROM AN_FS_DATA WHERE path = ?</fsDeleteDataQuery>
    <fsUpdateDataPathQuery>UPDATE AN_FS_DATA SET path = ? WHERE path =
?</fsUpdateDataPathQuery>
    <fsFileLengthRetrievalQuery>SELECT length FROM AN_FS_PATH WHERE path =

```

```

?</fsFileLengthRetrievalQuery>
    <fsInsertPathQuery>INSERT INTO AN_FS_PATH
(path,is_directory,length,parent_path) VALUES (?, ?, ?, ?)</fsInsertPathQuery>
    <fsListFilesQuery>SELECT path FROM AN_FS_PATH WHERE parent_path =
?</fsListFilesQuery>
    <fsPathRetrievalQuery>SELECT * FROM AN_FS_PATH WHERE path =
?</fsPathRetrievalQuery>
    <fsReadDataChunkQuery>SELECT data FROM AN_FS_DATA WHERE path = ? AND sequence
= ?</fsReadDataChunkQuery>
    <fsSetFileLengthQuery>UPDATE AN_FS_PATH SET length = ? WHERE path =
?</fsSetFileLengthQuery>
    <fsTableInitQueries>
        <query>CREATE TABLE AN_FS_PATH (path VARCHAR(256), is_directory BIT,
length BIGINT, parent_path VARCHAR(256), PRIMARY KEY(path), FOREIGN KEY (parent_path)
REFERENCES AN_FS_PATH(path))</query>
        <query>CREATE TABLE AN_FS_DATA (path VARCHAR(256), sequence BIGINT, data
VARBINARY(max), PRIMARY KEY (path,sequence), FOREIGN KEY (path) REFERENCES
AN_FS_PATH(path) ON DELETE CASCADE)</query>
        <query>CREATE INDEX index_parent_id ON AN_FS_PATH(parent_path)</query>
    </fsTableInitQueries>
    <fsTablesCheckQuery>SELECT path FROM AN_FS_PATH WHERE path =
'/'</fsTablesCheckQuery>
    <fsUpdateDataChunkQuery>UPDATE AN_FS_DATA SET data = ? WHERE path = ? AND
sequence = ?</fsUpdateDataChunkQuery>
    <fsWriteDataChunkQuery>INSERT INTO AN_FS_DATA (path,sequence,data) VALUES
(?, ?, ?)</fsWriteDataChunkQuery>
    <!--fsMergeDataChunkQuery>MERGE AN_FS_DATA AS dest USING (SELECT ?, ?, ?) AS
src (path, sequence, data) ON (dest.path = src.path AND dest.sequence = src.sequence)

```

```

WHEN MATCHED THEN UPDATE SET data = src.data WHEN NOT MATCHED THEN INSERT(path,
sequence, data) VALUES (src.path, src.sequence, src.data);</fsMergeDataChunkQuery-->
</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>
    <fsDataChunkSize>10240</fsDataChunkSize>
    <fsDeletePathQuery>DELETE FROM AN_FS_PATH WHERE path = ?</fsDeletePathQuery>
    <fsDeleteDataQuery>DELETE FROM AN_FS_DATA WHERE path = ?</fsDeleteDataQuery>
    <fsUpdateDataPathQuery>UPDATE AN_FS_DATA SET path = ? WHERE path =
?</fsUpdateDataPathQuery>
    <fsFileLengthRetrievalQuery>SELECT length FROM AN_FS_PATH WHERE path =
?</fsFileLengthRetrievalQuery>
    <fsInsertPathQuery>INSERT INTO AN_FS_PATH
(path, is_directory, length, parent_path) VALUES (?, ?, ?, ?)</fsInsertPathQuery>
    <fsListFilesQuery>SELECT path FROM AN_FS_PATH WHERE parent_path =
?</fsListFilesQuery>
    <fsPathRetrievalQuery>SELECT * FROM AN_FS_PATH WHERE path =

```

```
?</fsPathRetrievalQuery>
    <fsReadDataChunkQuery>SELECT data FROM AN_FS_DATA WHERE path = ? AND sequence
= ?</fsReadDataChunkQuery>
    <fsSetFileLengthQuery>UPDATE AN_FS_PATH SET length = ? WHERE path =
?</fsSetFileLengthQuery>
    <fsTableInitQueries>
        <query>CREATE TABLE AN_FS_PATH (path VARCHAR(256), is_directory BOOLEAN,
length BIGINT, parent_path VARCHAR(256), PRIMARY KEY(path))</query>
        <query>CREATE TABLE AN_FS_DATA (path VARCHAR(256), sequence BIGINT, data
BYTEA, PRIMARY KEY (path,sequence))</query>
        <query>CREATE INDEX index_parent_id ON AN_FS_PATH(parent_path)</query>
    </fsTableInitQueries>
    <fsTablesCheckQuery>SELECT path FROM AN_FS_PATH WHERE path =
'/'</fsTablesCheckQuery>
    <fsUpdateDataChunkQuery>UPDATE AN_FS_DATA SET data = ? WHERE path = ? AND
sequence = ?</fsUpdateDataChunkQuery>
    <fsWriteDataChunkQuery>INSERT INTO AN_FS_DATA (path,sequence,data) VALUES
(?, ?, ?)</fsWriteDataChunkQuery>
```

```
<!--fsMergeDataChunkQuery>INSERT INTO AN_FS_DATA (path,sequence,data) VALUES
(?, ?, ?) ON CONFLICT DO UPDATE data=VALUES(data)</fsMergeDataChunkQuery-->
</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>
    <fsDataChunkSize>10240</fsDataChunkSize>
    <fsDeletePathQuery>DELETE FROM AN_FS_PATH WHERE path = ?</fsDeletePathQuery>
    <fsDeleteDataQuery>DELETE FROM AN_FS_DATA WHERE path = ?</fsDeleteDataQuery>
    <fsUpdateDataPathQuery>UPDATE AN_FS_DATA SET path = ? WHERE path =
?</fsUpdateDataPathQuery>
    <fsFileLengthRetrievalQuery>SELECT length FROM AN_FS_PATH WHERE path =
?</fsFileLengthRetrievalQuery>
    <fsInsertPathQuery>INSERT INTO AN_FS_PATH
(path, is_directory, length, parent_path) VALUES (?, ?, ?, ?)</fsInsertPathQuery>
    <fsListFilesQuery>SELECT path FROM AN_FS_PATH WHERE parent_path =
```

```

?</fsListFilesQuery>
    <fsPathRetrievalQuery>SELECT * FROM AN_FS_PATH WHERE path =
?</fsPathRetrievalQuery>
    <fsReadDataChunkQuery>SELECT data FROM AN_FS_DATA WHERE path = ? AND sequence
= ?</fsReadDataChunkQuery>
    <fsSetFileLengthQuery>UPDATE AN_FS_PATH SET length = ? WHERE path =
?</fsSetFileLengthQuery>
    <fsTableInitQueries>
        <query>CREATE TABLE AN_FS_PATH (path VARCHAR(256) NOT NULL, is_directory
SMALLINT, length BIGINT, parent_path VARCHAR(256), PRIMARY KEY(path))</query>
        <query>CREATE TABLE AN_FS_DATA (path VARCHAR(256) NOT NULL, sequence
BIGINT NOT NULL, data BLOB(2G) NOT LOGGED, PRIMARY KEY (path,sequence))</query>
        <query>CREATE INDEX index_parent_id ON AN_FS_PATH(parent_path)</query>
    </fsTableInitQueries>
    <fsTablesCheckQuery>SELECT path FROM AN_FS_PATH WHERE path =
'/'</fsTablesCheckQuery>
    <fsUpdateDataChunkQuery>UPDATE AN_FS_DATA SET data = ? WHERE path = ? AND
sequence = ?</fsUpdateDataChunkQuery>
    <fsWriteDataChunkQuery>INSERT INTO AN_FS_DATA (path,sequence,data) VALUES
(?, ?, ?)</fsWriteDataChunkQuery>
    <fsMergeDataChunkQuery>MERGE INTO AN_FS_DATA AS dest USING (VALUES(?, ?, ?))
AS src (path, sequence, data) ON (dest.path = src.path AND dest.sequence =
src.sequence) WHEN MATCHED THEN UPDATE SET dest.data = src.data WHEN NOT MATCHED THEN

```

```
INSERT(path, sequence, data) VALUES (src.path, src.sequence,
src.data)</fsMergeDataChunkQuery>
</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/file system 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.
fsDataChunkSize	The maximum size for a chunk of data stored in the file system.
fsDeletePathQuery	The query template to delete all the files in a specific path in the file system.
fsDeleteDataQuery	The query template to delete selected data from the file system.
fsFileLengthRetrievalQuery	The query template to retrieve the length of a file stored in the file system.
fsInsertPathQuery	The query template to insert a new path to the file system.
fsListFilesQuery	The query template to list all the files in the file system.
fsPathRetrievalQuery	The query template to retrieve all the paths available in the file system.
fsReadDataChunkQuery	The query template to read a data chunk saved in the file system.
fsSetFileLengthQuery	The query template to specify a length for the files saved in the file path.
fsTableInitQueries	The query template to initiate a table saved in the file system.
fsTablesCheckQuery	The query template to check the tables saved in the file system.
fsMergeDataChunkQuery	The query template to merge two or more data chunks saved in the file system.
fsUpdateDataChunkQuery	The query template to update a data chunk saved in the file system.
fsWriteDataChunkQuery	The query template to create a new data chunk in the file system.

Configuring the datasources

You need to define datasources to connect to the above underlying RDBMS implementations as described below.
Configuring the datasource for the Analytics Record Store

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](#).

Configuring the datasource for the Analytics File System datasource

Change the configurations of the `WSO2_ANALYTICS_FS_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	<p>This is the default mode.</p> <p>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.</p>

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.	<code>http://localhost:9763</code>
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/: This parameter is com
TrustStorePassword	The password to access the trust store.	wso2carbon This parameter is com

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](#)

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 REST API](#)
- [Publishing Data Using Java Client Through Thrift/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](#)

Event Stream Simulator

Send multiple events

Input Data by File		
File	Stream Configuration	Action
No file has been uploaded		
<input type="button" value="Browse..."/> No file selected.	<input type="button" value="upload"/>	

Send single event

Event Stream Name *	<input type="button" value="select event stream"/>
<input type="button" value="Send"/>	

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 *	<input type="button" value="org.wso2.event.statistics.stream:1.0.0"/>																			
Stream Attributes <table border="1"> <tr> <td>Meta Attributes</td> </tr> <tr> <td>timestamp(<i>long</i>) *</td> <td><input type="text"/></td> </tr> <tr> <td>isPowerSaverEnabled(<i>bool</i>) *</td> <td><input type="text"/></td> </tr> <tr> <td>sensorId(<i>int</i>) *</td> <td><input type="text"/></td> </tr> <tr> <td>sensorName(<i>string</i>) *</td> <td><input type="text"/></td> </tr> <tr> <td>Correlation Attributes</td> </tr> <tr> <td>longitude(<i>double</i>) *</td> <td><input type="text"/></td> </tr> <tr> <td>latitude(<i>double</i>) *</td> <td><input type="text"/></td> </tr> <tr> <td>Payload Attributes</td> </tr> <tr> <td>humidity(<i>float</i>) *</td> <td><input type="text"/></td> </tr> <tr> <td>sensorValue(<i>double</i>) *</td> <td><input type="text"/></td> </tr> </table>		Meta Attributes	timestamp(<i>long</i>) *	<input type="text"/>	isPowerSaverEnabled(<i>bool</i>) *	<input type="text"/>	sensorId(<i>int</i>) *	<input type="text"/>	sensorName(<i>string</i>) *	<input type="text"/>	Correlation Attributes	longitude(<i>double</i>) *	<input type="text"/>	latitude(<i>double</i>) *	<input type="text"/>	Payload Attributes	humidity(<i>float</i>) *	<input type="text"/>	sensorValue(<i>double</i>) *	<input type="text"/>
Meta Attributes																				
timestamp(<i>long</i>) *	<input type="text"/>																			
isPowerSaverEnabled(<i>bool</i>) *	<input type="text"/>																			
sensorId(<i>int</i>) *	<input type="text"/>																			
sensorName(<i>string</i>) *	<input type="text"/>																			
Correlation Attributes																				
longitude(<i>double</i>) *	<input type="text"/>																			
latitude(<i>double</i>) *	<input type="text"/>																			
Payload Attributes																				
humidity(<i>float</i>) *	<input type="text"/>																			
sensorValue(<i>double</i>) *	<input type="text"/>																			
<input type="button" value="Send"/>																				

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

5. Click **Send**, to send the events to the event stream.

Sending Multiple Events

You can send multiple events to a defined event stream via the event simulator as described in the following

sections.

- Sending Multiple Events Using a CSV File
- Sending Multiple Events Using a Datasource

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.

1. Log in to the product management console using the following URL: https://<PRODUCT_HOST>:<PRODUCT_PORT>/carbon/
2. Click **Tools**, and then click **Event Simulator**.
3. Select the **Event Stream Name** from the drop down list.
4. Browse and upload the CSV file. It will be hot deployed in the server. Refresh the page to view the uploaded file.
5. Click **Configure**, to configure the CSV file to specify the field delimiter before simulating the event flow as shown below.

[Send multiple events](#)

Input Data by File 		Stream Configuration	Delay between events(ms)	Action
File	events.csv	click the configure button	click the configure button	 
<input type="button" value="Choose File"/> No file chosen <input type="button" value="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*	<input type="button" value="test:1.0.0"/>
Field delimiter*	<input type="text"/>
Delay between events in milliseconds*	<input type="text" value="1000"/>
<input type="button" value="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 		Stream Configuration	Delay between events(ms)	Action
File	events.csv	test:1.0.0	click the configure button	  
<input type="button" value="Choose File"/> No file chosen <input type="button" value="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
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

Input Data by Data Source		switch to upload file for simulation										
Name	Data Source Name	Table Name	Column names	Stream ID								
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 border="1"> <thead> <tr> <th colspan="2">Table column name</th> </tr> </thead> <tbody> <tr> <td>Payload Attributes</td> <td></td> </tr> <tr> <td>attrib1(<i>int</i>) *</td> <td>s4</td> </tr> <tr> <td>attrib2(<i>string</i>) *</td> <td>hostname</td> </tr> </tbody> </table>					Table column name		Payload Attributes		attrib1(<i>int</i>) *	s4	attrib2(<i>string</i>) *	hostname
Table column name												
Payload Attributes												
attrib1(<i>int</i>) *	s4											
attrib2(<i>string</i>) *	hostname											
Test Connection Save												

7. Click **Test Connection**, and then click **Save**, to test the connection and save it. When the configuration file gets hot deployed, click **Play** to simulate sending events via the database as shown below.

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

Publishing Data Using Java Client Through REST API

For information on how to publish data to WSO2 DAS using a Java Client through REST API operations, see [Analytics REST API Guide](#).

Publishing Data Using Java Client Through Thrift/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</groupId>
    <artifactId>org.wso2.carbon.databridge.agent</artifactId>
    <version>${carbon.commons.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.carbon</groupId>
    <artifactId>org.wso2.carbon.databridge.commons</artifactId>
    <version>${carbon.commons.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.carbon</groupId>
    <artifactId>org.wso2.carbon.databridge.commons.thrift</artifactId>
    <version>${carbon.commons.version}</version>
</dependency>
```

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>
            <TrustStore>src/main/resources/client-truststore.jks</TrustStore>
            <TrustStorePassword>wso2carbon</TrustStorePassword>
            <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>
            <TrustStore>src/main/resources/client-truststore.jks</TrustStore>
            <TrustStorePassword>wso2carbon</TrustStorePassword>
            <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);`

i 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);
```

-  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.

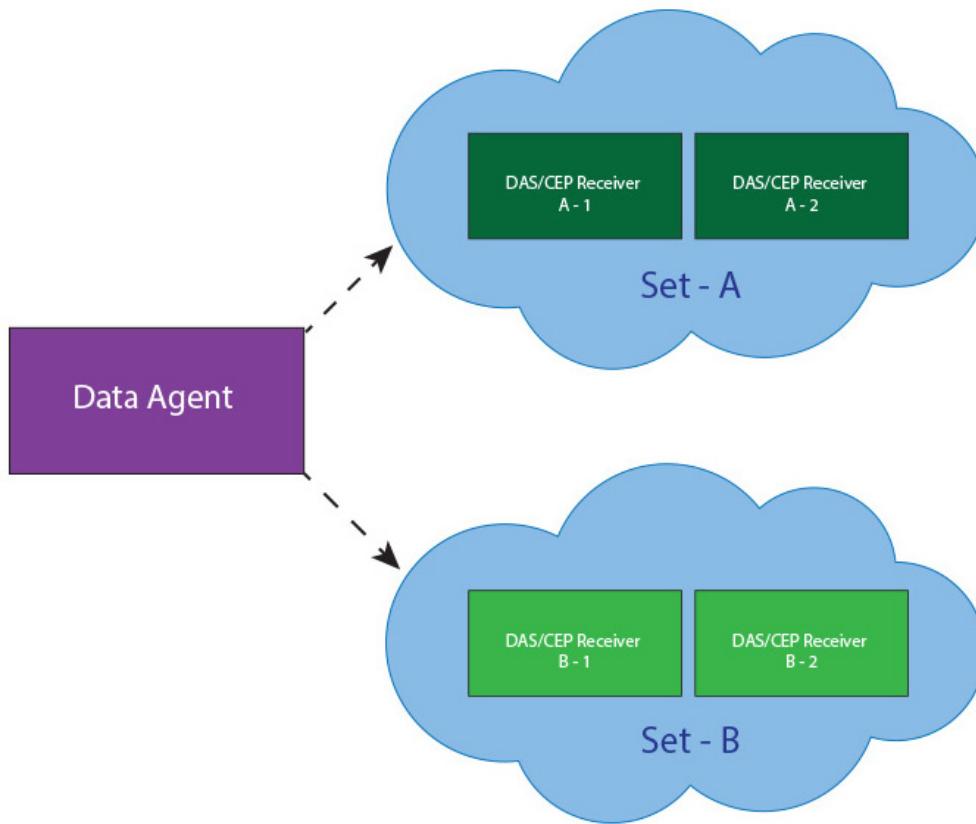
-  All the scenarios described below are different ways to use Data Agents with multiple receivers using the load balancing functionality. Each approach has its own advantages. Select the most appropriate scenario depending on your requirements.

- 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.

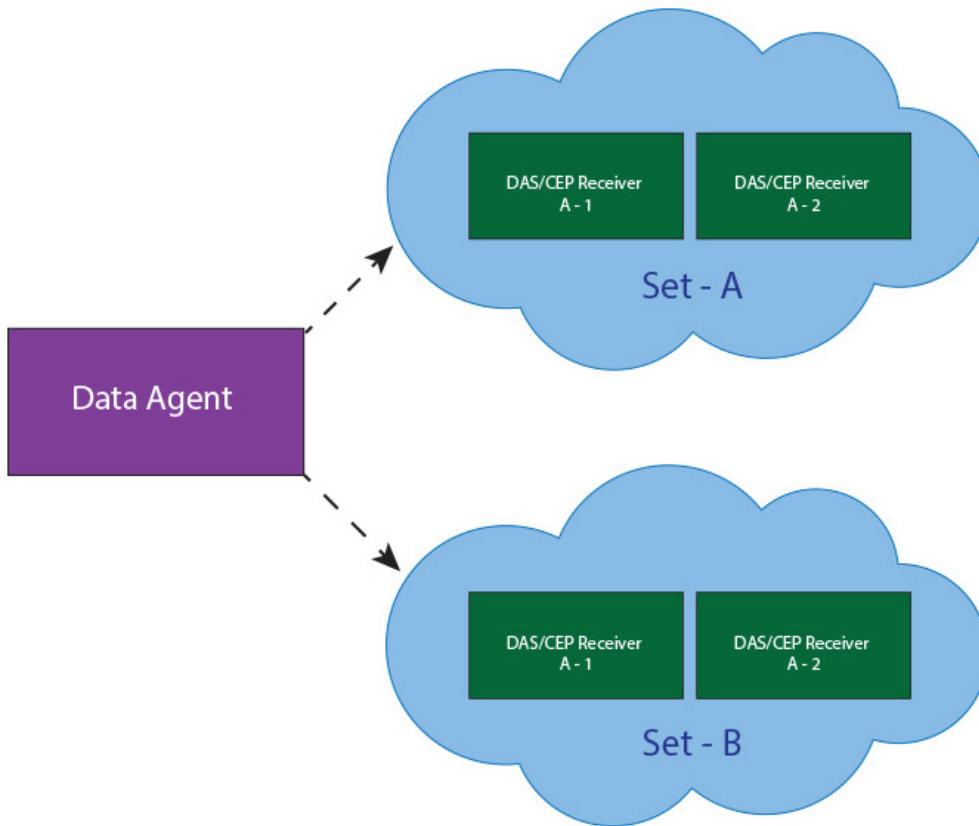
For this functionality, include the server URL in the Data Agent as a general DAS/CEP receiver URL. The URL should be entered in a comma separated format as shown below.

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

- (i) 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.

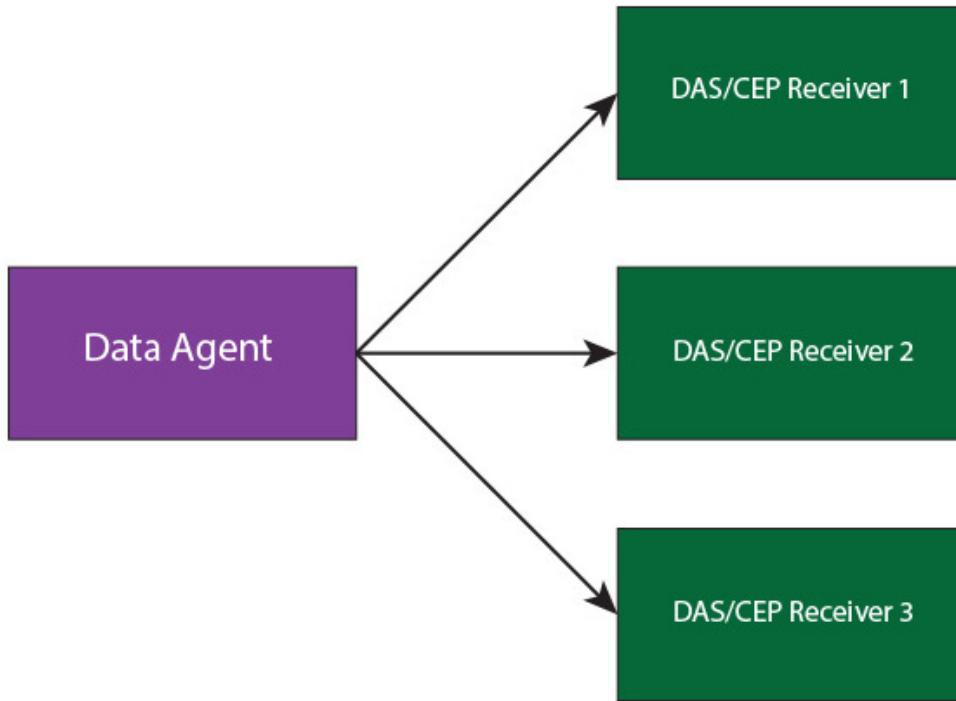
Similar to the other scenarios, you can describe this as a receiver URL. The sets should be mentioned within curly braces separated by commas. Further more, each receiver that belongs to the set, should be within the curly braces and with the receiver URLs in a comma separated format. The receiver URL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

i <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.

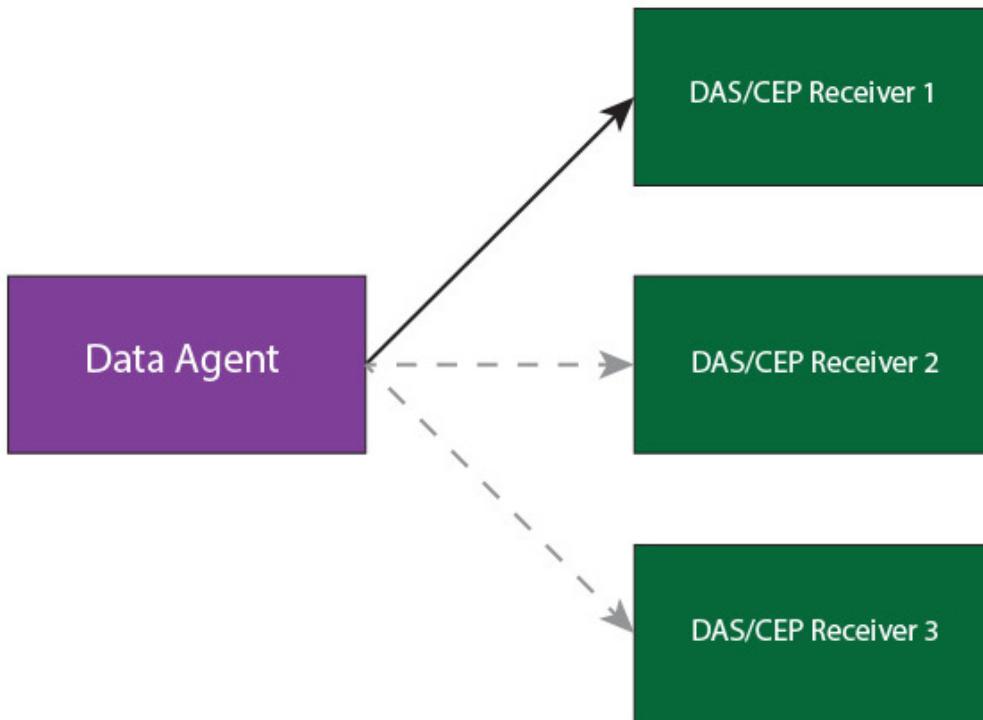
Similar to [load balancing between a set of servers](#), in this scenario you need to modify the Data Agent URL. You should include all DAS/CEP receiver URLs within curly braces ({}) separated with commas as shown below.

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

i <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
  
```

i 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](#).

i 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.

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

Adding the default JMX profile

You can set up the default JMX toolbox which is shipped with WSO2 DAS to monitor system resources of a WSO2 server (CPU/memory/OS) running on Linux. Follow the steps below to setup this default JMX toolbox.

1. Log in to the management console using admin/admin credentials and the following URL: `https://<DAS_HOST>:<DAS_PORT>/carbon/`
2. Click **Configure**, and then click **JMX Agent**.
3. Click **Add Default JMX Toolbox to monitor system resources of a WSO2 server running on Linux (CPU/Memory/OS)**. This adds a pre-configured server profile to monitor the JMX attributes of WSO2 DAS itself as shown below.

Home > Configure > JMX Agent

JMX Monitoring Profiles

[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:

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=ClassLoader	LoadedClassCount	loadedClassCount	
java.lang:type=Threading	ThreadCount	threadCount	
java.lang:type=Threading	PeakThreadCount	peakThreadCount	

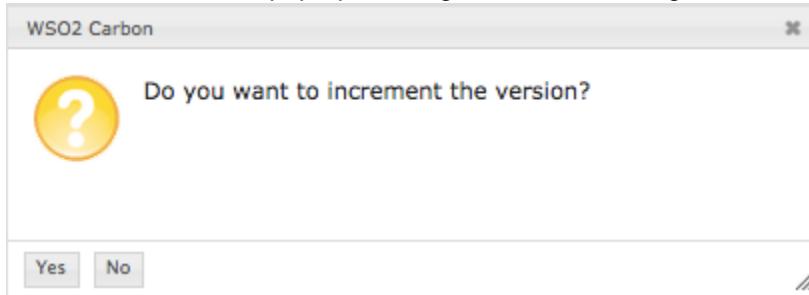
7. Click an MBean on the list that loads to view its attributes list.
8. Select the attributes that you require to monitor by this profile as shown below. You can set an alias to easily identify the data in the Data Access Layer of WSO2 DAS.

Select attributes:

Selected Attributes:

MBean	Attribute	Alias	Actions
CatalinaJ2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputwebsocket,j2eeType=Servlet,name=default	eventProvider	CatalinaJ2EEApplication=none,J2EEServer=none,WebModule=/localhost	
CatalinaJ2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputwebsocket,j2eeType=Servlet,name=default	maxTime	CatalinaJ2EEApplication=none,J2EEServer=none,WebModule=/localhost	

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.

[Load MBeans](#)

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

are described below.

The details you enter in the above screen

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.	<pre>service:jmx:rmi://localhost:1111/jndi/rmi://localhost:9999</pre> <p>(Use this example to monitor DAS by itself).</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Change all occurrences of the host and port (if you have set a port offset server) accordingly in the JMX server URL. </div>
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/analytics,j2eeType=Filter,name=Tomcat WebSocket (JSR356) Filter
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/analytics,j2eeType=Servlet,name=cxf
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/analytics,j2eeType=Servlet,name=default
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/analytics,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/outputui,j2eeType=Filter,name=Tomcat WebSocket (JSR356) Filter
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputui,j2eeType=Servlet,name=JAXServlet
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputui,j2eeType=Servlet,name=default
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputui,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	Remove
Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost/outputwebsocket,j2eeType=Servlet,name=default	maxTime	Catalina:J2EEApplication=none,J2EEServer=none,WebModule=/localhost	Remove

Save**Cancel**

8. Click **Save**. You view the new server profile added to the list of existing profiles as shown below.



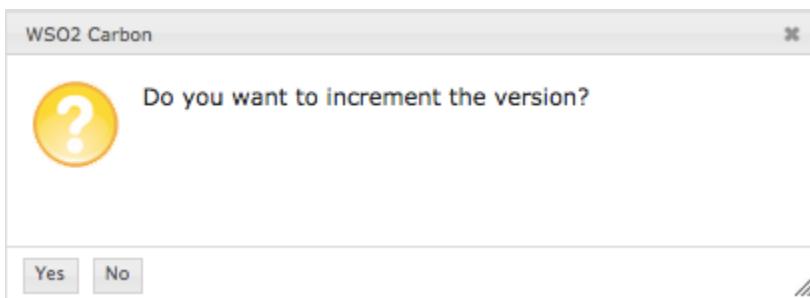
You can enable/disable monitoring of JMX attributes, and also edit or delete the monitoring profiles using the options provided in this screen.

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	Disable Edit Delete



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

Uploading the C-App

WSO2 DAS is shipped with a sample C-App for the JMX Agent. This includes all the artifacts which you need to publish data to the JMX agent which you enabled above, and to persist that data. Follow the steps below to upload this sample Carbon Application (c-App) file to the DAS. For more information, see [Carbon Application Deployment for DAS](#).

1. Log in to the DAS management console using the following

URL: https://<DAS_HOST>:<DAS_PORT>/carbon/

2. Click **Main**, and then click **Add** in the **Carbon Applications** menu.
3. Click **Choose File**, and upload the <DAS_HOME>/samples/capps/JMX_Agent.car file as shown below.

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

Add Carbon Applications

Upload Carbon Application

Carbon Application Artifact(.car) JMX_Agent.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_Realtimme_Analytics	1.0.0	Delete Download
JMX_Agent_CApp	1.0.0	Delete Download

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*

Search By Date Range By Query

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_commt
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 **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.

Home > Manage > Event > Streams Help

Edit Event Stream

Enter Event Stream Details																													
<input checked="" type="checkbox"/> Persist Event Stream Record Store <input type="button" value="EVENT_STORE"/>																													
Meta Data Attributes <table border="1"> <tr> <td><input type="checkbox"/> Persist Attribute</td> <td>Attribute Name</td> <td>Attribute Type</td> <td>Primary Key</td> <td>Index Column</td> <td>Score Param</td> </tr> <tr> <td><input type="checkbox"/></td> <td>subnet-mask</td> <td>INTEGER</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table>						<input type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	<input type="checkbox"/>	subnet-mask	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																								
<input type="checkbox"/>	subnet-mask	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																								
Correlation Data Attributes <table border="1"> <tr> <td><input type="checkbox"/> Persist Attribute</td> <td>Attribute Name</td> <td>Attribute Type</td> <td>Primary Key</td> <td>Index Column</td> <td>Score Param</td> </tr> <tr> <td><input type="checkbox"/></td> <td>network-ip</td> <td>INTEGER</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table>						<input type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	<input type="checkbox"/>	network-ip	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																								
<input type="checkbox"/>	network-ip	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																								
Payload Data Attributes <table border="1"> <tr> <td><input checked="" type="checkbox"/> Persist Attribute</td> <td>Attribute Name</td> <td>Attribute Type</td> <td>Primary Key</td> <td>Index Column</td> <td>Score Param</td> </tr> <tr> <td><input checked="" type="checkbox"/></td> <td>hostname</td> <td>FACET</td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td><input checked="" type="checkbox"/></td> <td>ip-address</td> <td>INTEGER</td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td><input checked="" type="checkbox"/></td> <td>state</td> <td>BOOLEAN</td> <td><input type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </table>						<input checked="" type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param	<input checked="" type="checkbox"/>	hostname	FACET	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ip-address	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	state	BOOLEAN	<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																								
<input checked="" type="checkbox"/>	hostname	FACET	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>																								
<input checked="" type="checkbox"/>	ip-address	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>																								
<input checked="" type="checkbox"/>	state	BOOLEAN	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>																								
Arbitrary Data Attributes <i>No arbitrary data attributes are defined</i>																													
Attribute Name : <input type="text"/> Attribute Type : <input type="button" value="INTEGER"/> Primary Key : <input type="checkbox"/> Index Column : <input type="checkbox"/> Score Param : <input type="checkbox"/> <input type="button" value="Add"/>																													

[Back](#) [Save Event Stream](#)

- Select **Persist Attribute**, if you want to persist a particular attribute.
- Select FACET for **Attribute Type**, if you want to persist an attribute as a facet. For more information on facets, see [Searching Data By Categories](#).
- 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 Data By Categories](#).

- Define and add any **Arbitrary Data Attributes** which you want to persist.

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

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
- Creating Custom Transports to Receive Events

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"/> (?) This depends on how events are published to the server, 'true' only if multiple receiver URLs are defined in different receiver groups ({}).	
To Event Stream* <input type="text" value="testEventStream:1.0.0"/> (?) The event stream that will be generated by the received events	
Mapping Configuration Message Format* <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.

i To create an event receiver via the management console, you must have at least one event stream defined.

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	

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.

- 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.

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

- Log in to the management console, and click **Main**.
- Click **Receivers** in the **Event** menu. You view the **Available Event Receivers** list.
- Click the **Edit** button of the corresponding event receiver to edit it. This opens **Edit Event Receiver Configurations** XML editor.
- 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.

- Uncomment the following line in the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file, to enable the mail transport receiver in the Axis2 level:
`<transportReceiver name="mailto" class="org.apache.axis2.transport.mail.MailTransportListener"/>`
- 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"/>	

i 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="EVENT-RECEIVER-NAME" 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 in WSO2 CEP Documentation.

- 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
<code>events.duplicated.in.cluster</code>	If this property is set to <code>true</code> , events received by <code>file-tail</code> receivers are re-created in every node in the cluster.	Boolean	<code>false</code>

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*	FileTailInputEventAdapter ⑦ Enter a unique name to identify Event Receiver
From	file-tail ⑦ Select the type of Adapter to receive events
Adapter Properties	
File path*	/Users/Desktop/abc.txt ⑦ Absolute path of the file (Eg: /home/cep/cep_3.1.0/wso2cep-3.1.0/repository/logs/wso2carbon.log)
Delay	10 ⑦ The delay between checks for new content on file in milliseconds.
Start From End*	true ⑦ Set to true to tail from the end of the file, false to tail from the beginning of the file.
To	testEventStream:1.0.0 ⑦ The event stream that will be generated by the received events
Mapping Configuration	
Message Format*	text ⑦ Select the input message format
Advanced	
<input type="button" value="Add Event Receiver"/>	

i 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="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 in WSO2 CEP Documentation.

- [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 `<DAS_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.

 Help

Create a New Event Receiver

Enter Event Receiver Details

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

Add Event Receiver

After entering the transport type in adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events . Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see [Input Mapping Types](#) .

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

```
<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 in WSO2 CEP Documentation.

- 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
- 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.

i 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.
 - <ACTIVEMQ_HOME>/lib/activemq-core-xxx.jar
 - <ACTIVEMQ_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"/> <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="jms"/> <div style="font-size: small; margin-top: -10px;">⑦ Select the type of Adapter to receive events</div>
Adapter Properties	
Topic/Queue Name* <input type="text" value="Test Topic"/> <div style="font-size: small; margin-top: -10px;">⑦ Topic/Queue name of the input stream</div>	JNDI Initial Context Factory Class* <input type="text" value="org.apache.activemq.jndi.ActiveMQInitialContextFactory"/> <div style="font-size: small; margin-top: -10px;">⑦ JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</div>
JNDI Provider URL* <input type="text" value="tcp://localhost:61616"/> <div style="font-size: small; margin-top: -10px;">⑦ URL of the JNDI provider.</div>	The JMS connection password <input type="text"/>
The JMS connection username <input type="text"/>	Connection Factory JNDI Name* <input type="text" value="TopicConnectionFactory"/> <div style="font-size: small; margin-top: -10px;">⑦ The JNDI name of the connection factory.</div>
Destination Type* <input type="text" value="topic"/> <div style="font-size: small; margin-top: -10px;">⑦ Type of the destination.</div>	Enable Durable Subscription <input type="text" value="true"/> <div style="font-size: small; margin-top: -10px;">⑦ Whether the subscription is durable or not.</div>
Durable Subscriber Client ID <input type="text" value="wso2dasclient1"/> <div style="font-size: small; margin-top: -10px;">⑦ Name/ID of the durable subscriber (If any value added, Durable subscription will be enabled).</div>	
JMS Properties <input type="text"/> <div style="font-size: small; margin-top: -10px;">⑦ Axis2 JMS Properties, e.g. "property1: value1, property2: value2"</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	

i After entering the above adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events . Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see [Input Mapping Types](#) .

You can also define the respective adapter properties of the event receiver based on the transport type within the <from> element of the event receiver configuration in the <PRODUCT_HOME>/repository/deployment/server/eventreceivers/ directory as follows.

```

<eventReceiver name="ActiveMQJMSInputEventAdapter" statistics="disable"
    trace="disable" xmlns="http://wso2.org/carbon/eventreceiver">
    <from eventAdapterType="jms">
        <property
name="java.naming.factory.initial">org.apache.activemq.jndi.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.ConnectionFactoryJND
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.DurableSubscriberCli
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 in WSO2 CEP Documentation.

- [Sample 0009 - Receiving Map Events via JMS Transport - ActiveMQ](#)
- [Sample 0010 - Receiving Custom Map Events via JMS Transport - ActiveMQ](#)
- [Sample 0011 - Receiving JSON, Text, XML Events via JMS Transport - ActiveMQ](#)
- [Sample 0021 - Receiving Map Events via JMS Transport - ActiveMQ \(For Queue\)](#)

IBM WebSphere MQ JMS Event Receiver

IBM WebSphere MQ JMS event receiver is an internal event receiver that comes with WSO2 products by default . You can configure it with **XML**, **map**, **JSON**, and **text** input mapping types.

- [Prerequisites](#)
- [Creating an IBM WebSphere MQ JMS event receiver](#)

Prerequisites

Follow the instructions below to complete the prerequisites before starting the event receiver configurations.
Configuring WebSphere MQ

Follow the steps below to configure WebSphere MQ.

Configuring JMSAdmin.conf file

1. Download and install [WebSphere MQ pack with the latest fixes](#). For more information on installing, go to the [IBM documentation](#).
2. Start the IBM WebSphere MQ JMS server.
3. Open the <WebSphere_MQ_HOME>/java/bin/JMSAdmin.config file in a text editor.
4. Comment the existing INITIAL_CONTEXT_FACTORY, and add an INITIAL_CONTEXT_FACTORY named com.sun.jndi.fscontext.RefFSContextFactory as follows.

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
```

5. Comment the default PROVIDER_URL, and use a directory path instead. Ensure the directory is created in the file system (e.g., C:/JNDI-Directory).

 If there are .bindings files of earlier versions already existing in this directory, delete them. It should typically be an empty folder.

Your JMSAdmin.config file should now look similar to this:

```

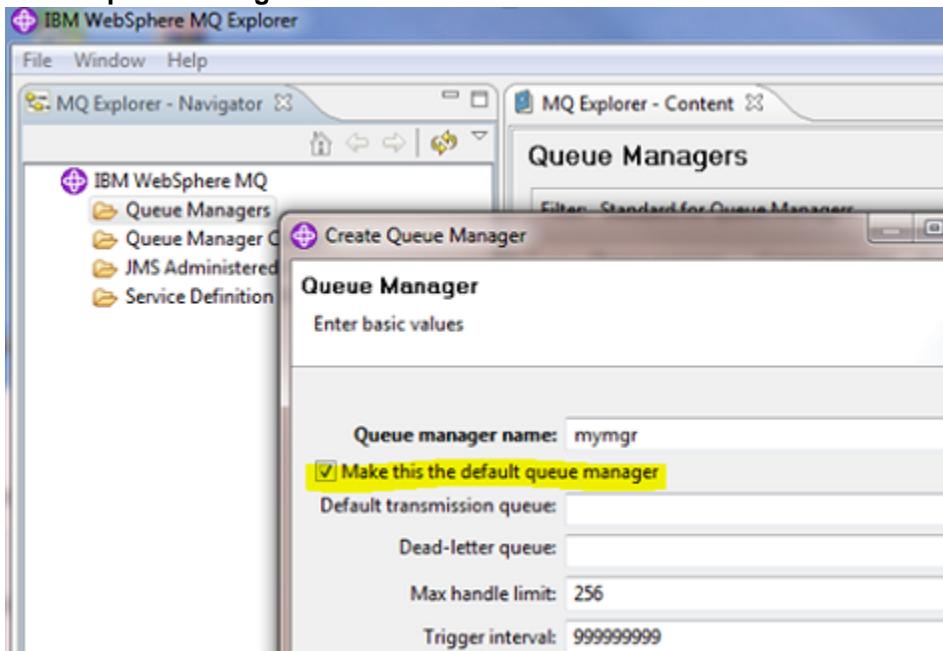
# appropriate one should be uncommented.
#
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WMQInitialContextFactory
#
# The following line specifies the URL of the service provider's initial
# context. It currently refers to an LDAP root context. Examples of a
# file system URL and WebSphere's JNDI namespace are also shown, commented
# out.
#
#PROVIDER_URL=ldap://polaris/o=ibm,c=us
PROVIDER_URL=file:/C:/JNDI-Directory
#PROVIDER_URL=iiop://localhost/
#PROVIDER_URL=localhost:1414/SYSTEM.DEF.SVRCONN
.....

```

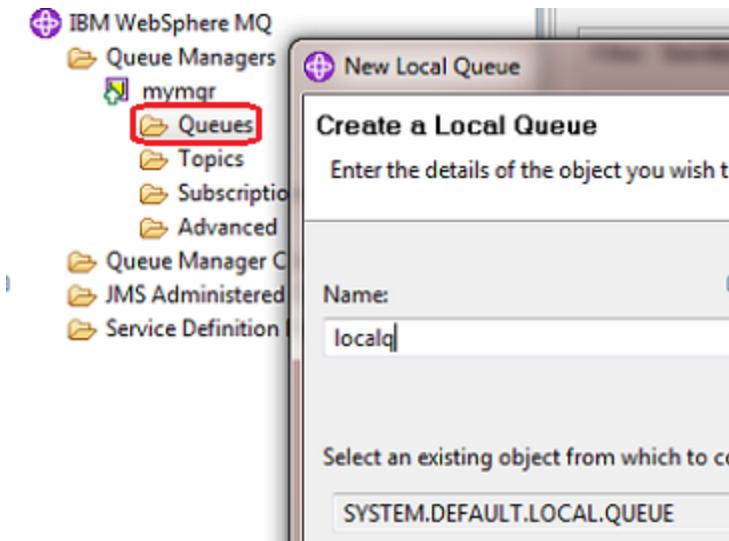
6. Restart the WebSphere MQ service.

Creating the Queue in WebSphere MQ

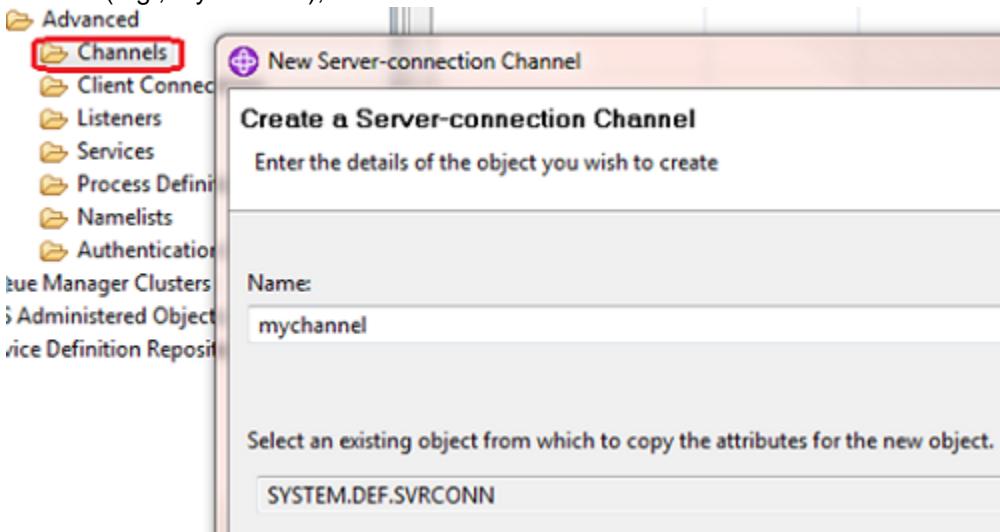
1. Start IBM WebSphere MQ Explorer, and create a new queue manager. Make sure you select **make this the default queue manager** check box and leave the default values on the other fields as shown below.



2. Select the options to **Start Queue Manager**, **Autostart Queue Manager**, and **Create server connection channel**, and then click **Next**.
3. Select the option to create a listener configuration for TCP/IP, and provide a port number (e.g., 1415).
4. Select the created queue manager and expand its navigation tree. Click **Queues** in the tree and create a new local queue (e.g., localq) as shown below.



5. Keep the default configurations, and click **Finish**.
6. Click **Topics** in the tree view, and create a new local topic (e.g., **localt**).
7. Right-click **Channels** under **Advanced**, and click **New Server-connection Channel**. Provide a name for the channel (e.g., **myChannel**), and click **Next**.



8. Set the transmission protocol as TCP, and click **Finish**.
A listener is created and is running on the given port (e.g., 1415). You should be able to view it by clicking the **listeners** icon.

Generating the .bindings file

1. Navigate to the <WebSphere_MQ_HOME> / java/bin/ directory, and invoke the IVT app by running the following command:

```
IVTRun.bat -nojndi -client -m mymgr -host localhost -channel mychannel
```

2. Create the default set of JNDI bindings by running the following command on the command prompt:

```
IVTSetup.bat
```

3. Execute the **IVTRun** tool as follows.

```
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 CEP

Follow the instructions below to configure WSO2 CEP.

1. If you set up WSO2 CEP on a different machine from WebSphere MQ, copy C:/JNDI-Directory/ to that machine. The bindings file allows you to access WebSphere queues from any machine in the network.
2. Copy the following JAR files from the <WebSphere_MQ_HOME>/java/lib/ directory to the <CEP_HOME>/repository/components/lib/ directory .
 - com.ibm.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 <CEP_HOME>/repository/components/dropins/ directory.
 - com.ibm.mq_2.0.0.jar
 - fscontext_1.0.0.jar
 5. Add the following files to the <CEP_HOME>/repository/components/lib directory.
 - jms.jar
 - jta.jar

6. Log in to the JMSAdmin tool, and create a queue named **bogusq** by running the following commands in JMSAdmin shell.

```
DEFINE Q(bogusq) QMGR(mymgr)
ALTER Q(bogusq) QUEUE(localq)
```



localq is the queue you created [earlier](#). You use two queues for the queue scenario. The queue named **bogusq** is the default destination since you need the default queue (ivtQ) for your proxy service only. If you use **ivtQ** here, all the services deployed in WSO2 CEP (XKMS, echo, wso2carbon-sts etc.) will start listening on the same queue.

7. Repeat these steps for the topic scenarios. For example:

```
DEFINE T(bogust)
ALTER T(bogust) TOPIC(localt)
```



localt is the topic you created [earlier](#).

8. Configure the `<CEP_HOME>\repository\conf\axis2\axis2.xml` file as follows:

```

<transportReceiver name="jms" class="org.apache.axis2.transport.jms.JMSListener">
    <parameter name="myTopicConnectionFactory" locked="false">
        <parameter name="java.naming.factory.initial"
locked="false">com.sun.jndi.fscontext.RefFSContextFactory</parameter>
            <parameter name="java.naming.provider.url"
locked="false">file:/C:/JNDI-Directory</parameter>
                <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">ivtQCF</parameter>
                    <parameter name="transport.jms.ConnectionFactoryType"
locked="false">topic</parameter>
                </parameter>

                <!--parameter name="SQProxyCF" locked="false">
                    <parameter
name="java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFactory</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>
                        </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	
Input Event Adapter Type*	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*	ivtQCF 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	
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"/>	

- ⑤ 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.ConnectionFactoryJND
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.DurableSubscriberCli
J M S Properties	Valid "property:value" pairs of Axis2 JMS properties (e.g. "property1: value1, property2: value2") For more information on Axis2 JMS properties, go to Apache AXIS2 Transports Documentation .	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 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.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>
<input type="button" value="Advanced"/>	

- i** 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.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 Qpid event receiver type, see the following sample in WSO2 CEP Documentation.

- [Sample 0012 - Receiving Map, XML Events via JMS Transport - Qpid](#)

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
 - <W SO2MB_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 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	

i 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.SubscriptionDurability">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 in WSO2 CEP Documentation.

- [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
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"/>	

i 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="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 ID Kafka	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 in WSO2 CEP Documentation.

- [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**, **J**

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](#)

i 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="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 in WSO2 CEP Documentation.

- [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"/> ⑦ Enter a unique name to identify Event Receiver	
From	
Input Event Adapter Type* <input type="text" value="soap"/> ⑦ 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"/> ⑦ The event stream that will be generated by the received events	
Mapping Configuration	
Message Format* <input type="text" value="xml"/> ⑦ Select the input message format	
+ Advanced	
<input type="button" value="Add Event Receiver"/>	

- i After entering the transport type in adapter properties, select the **Event Stream** to which you want to map the incoming events, and the **Message Format** which you want to apply on the receiving events. Also, click **Advanced** to define custom input mappings based on the **Message Format** you selected. For more information on custom input mapping types, see [Input Mapping Types](#).

You can also define the respective adapter properties of the event receiver based on the transport type within the `<from>` element of the event receiver configuration in the `<PRODUCT_HOME>/repository/deployment/server/eventreceivers/` directory as follows.

```

<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 in WSO2 CEP Documentation.

- [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.

- [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; margin-top: -10px;">⑦ Enter a unique name to identify Event Receiver</div>	From <input type="text" value="websocket"/> <div style="font-size: small; margin-top: -10px;">⑦ 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; margin-top: -10px;">⑦ 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; 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](#)

i 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="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 in WSO2 CEP Documentation.

- [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.

- [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;">② 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;">② Select the type of Adapter to receive events</div>	
Usage Tips <div style="margin-top: 10px;">Following url formats are used to receive events: ws://localhost:9764/inputwebsocket/<receiver_name> wss://localhost:9444/inputwebsocket/<receiver_name></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="xml"/> <div style="font-size: small; margin-top: -10px;">② Select the input message format</div>	
+ Advanced	

[Add Event Receiver](#)

Related samples

For more information on websocket-local event receiver type, see the following sample in WSO2 CEP Documentation.

- 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.

[Help](#)

Create a New Event Receiver

Enter Event Receiver Details

Event Receiver Name* <input type="text" value="WSO2EventReceiver"/> <small>⑦ Enter a unique name to identify Event Receiver</small>	From Input Event Adapter Type* <input type="text" value="wso2event"/> <small>⑦ Select the type of Adapter to receive events</small>
<p>Following url formats are used to receive events For load-balancing: use "," to separate values for multiple endpoints <small>Eg: {tcp://<hostname>:<port>},{tcp://<hostname>:<port>}, ...}</small></p> <p>For failover: use " " to separate multiple endpoints <small>Eg: {tcp://<hostname>:<port>}/{tcp://<hostname>:<port>} ...}</small></p> <p>For more than one cluster: use "{}" to separate multiple clusters <small>Eg: {tcp://<hostname>:<port>}/{tcp://<hostname>:<port>} {tcp://<hostname>:<port>} ...}</small></p> <p>Ports available for Thrift protocol – TCP port:7612 or SSL port:7712 Ports available for Binary protocol – TCP port:9612 or SSL port:9712</p>	
Adapter Properties	
Is events duplicated in cluster <input type="text" value="false"/> <small>⑦ This depends on how events are published to the server, 'true' only if multiple receiver URLs are defined in different receiver groups ({}).</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="wso2event"/> <small>⑦ Select the input message format</small>	
+ Advanced	

[Add Event Receiver](#)

- i** 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 **A dvanced** 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 samples in WSO2 CEP Documentation.

- 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 * sensor.stream

Input Event Version * 1.0.6

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

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*

[Advanced](#)

XML Mapping

XPath Prefixes		
Prefix	Namespace	Actions
sen	http://wso2.org	<input type="button" value="Delete"/>

Prefix : Namespace :

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:isPowerSeved"/>	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_sensordId"/>	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.

Mapping Configuration

Message Format*
json
 Select the input message format

+ Advanced

JSON Mapping

Available JSON Mappings			
JSONPath : <input type="text" value="\$.sensorData.time"/>	Mapped To : <input type="text" value="meta_timestamp"/>	Type : long	Default Value : <input type="text"/>
JSONPath : <input type="text" value="\$.sensorData.powerSaving"/>	Mapped To : <input type="text" value="meta_isPowerSaverEnabled"/>	Type : bool	Default Value : true
JSONPath : <input type="text" value="\$.sensorData.id"/>	Mapped To : <input type="text" value="meta_sensorId"/>	Type : int	Default Value : <input type="text"/>
JSONPath : <input type="text" value="\$.sensorData.name"/>	Mapped To : <input type="text" value="meta_sensorName"/>	Type : string	Default Value : ---
JSONPath : <input type="text" value="\$.sensorData.long"/>	Mapped To : <input type="text" value="correlation_longitude"/>	Type : double	Default Value : <input type="text"/>
JSONPath : <input type="text" value="\$.sensorData.lat"/>	Mapped To : <input type="text" value="correlation_latitude"/>	Type : double	Default Value : <input type="text"/>
JSONPath : <input type="text" value="\$.sensorData.humidity"/>	Mapped To : <input type="text" value="humidity"/>	Type : float	Default Value : <input type="text"/>
JSONPath : <input type="text" value="\$.sensorData.value"/>	Mapped To : <input type="text" value="sensorValue"/>	Type : 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="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		Actions
Regular Expression		
(\w+)\s(\w+)\s(\w+)\s(\w+)		Delete
time\s(\w+)		Delete
powerSaved\s(\w+)		Delete
value\s(\w+)\s(\w+)		Delete

Below this is a form for adding new mappings:

Regular Expression : * Add

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

At the bottom, there is a form for defining a new mapping:

Regular Expression : Mapped To : Type: Default Value: Add

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 ConfigurationMessage 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

Creating Custom Transports to Receive Events

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.getTailor().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);
    }
}
```

i 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.

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:

- [Interactive Analytics](#)
- [Batch Analytics Using Spark SQL](#)
- [Realtime Analytics Using Siddhi](#)
- [Predictive Analytics Using WSO2 ML](#)

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.

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
<input type="checkbox"/>	subnet-mask	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Correlation Data Attributes					
Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
<input type="checkbox"/>	network-ip	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Payload Data Attributes					
Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
<input checked="" type="checkbox"/>	hostname	FACET	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	ip-address	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	state	BOOLEAN	<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 :

- Select **Persist Attribute**, if you want to persist a particular attribute.
- Select FACET for **Attribute Type**, if you want to persist an attribute as a facet. For more information on facets, see [Searching Data By Categories](#).
- 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 Data By Categories](#).
- Define and add any **Arbitrary Data Attributes** which you want to persist.

7. 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 **Attribute Type** parameter should be **FACET**.

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	Not required
Performing a drill down search	Required	Required	Required	Required	Required
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 Data By Categories](#). [Searching Data By Categories](#)

Facets and score parameters are used to categorize data based on attributes of an event stream in WSO2 DAS.

- Facets
- Score parameters

Facets

A facet is an attribute of indexed records which is used to classify the records by the attribute value. Facet attributes allow you to carry out a faceted extended search within the defined categories. There is no data type called FACET in event streams. Any STRING type field, of which you can define the attribute value as a JSON array can be indexed as a facet. Facets are defined when the table schema is created during the persisting of event streams.

Facets are used in the implementations of the following REST APIs.

- Drilling Down Through Categories via REST API
- Retrieving Specific Records through a Drill Down Search via REST API
- Retrieving the Number of Records Matching the Drill Down Criteria via REST API
- Retrieving the Event Count of Range Facets

Facet usage types

Facets are used in the Analytics REST API and Data Explorer in WSO2 DAS. Different usage types of facets are described below.

Searching by an attribute Extracting sub categories Performing a drill down search

Searching data within a value range

Searching by an attribute

This denotes implementing an attribute of a set of records as a facet. For an example, in a record which represents a book, you can define the AUTHOR field as a facet when you are persisting the event stream as shown below.

Enter Event Index Details

Persist Event Stream

Record Store
EVENT_STORE

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
<input checked="" type="checkbox"/>	AUTHOR	FACET	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	TITLE	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	PUBLISHED_DATE	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	PRICE	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	COUNT	INTEGER	<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 :

Click the **Simulate** option of the event stream to simulate sending events to the created event stream.

Available Event Streams

2 Event streams available

Event Stream Id	Event Stream Description	Actions
BOOK_STORE:1.0.0		<input type="button" value="Simulate"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
org.wso2.sample.httpd.logs:1.0.0	Sample of Httpd logs	<input type="button" value="Simulate"/>

For the attribute which you defined as a facet, you need to send its values as a JSON string array as shown in the example below .

Home > Manage > Event > Streams > List

Event Stream Simulator

Send single event

Event Stream Name *

BOOK_STORE:1.0.0

Stream Attributes

Payload Attributes

AUTHOR(string) *	[C.Dickens]
TITLE(string) *	A Tale of Two Cities
PUBLISHED_DATE(string) *	1986-08-09
COUNT(int) *	12
PRICE(float) *	30

Buttons: Send, Clear

You can use this facet to retrieve the records of the books which are written by a particular author using the [Data Explorer](#) as shown in the example below.

Home > Manage > Interactive Analytics > Data Explorer

[Help](#)

Data Explorer

Search

Table Name* BOOK_STORE Schedule Data Purging

Search By Date Range By Query

Search Query

AUTHOR C.Dickens Select a category Remove

Search Reset

Results

BOOK_STORE					
	AUTHOR	TITLE	PUBLISHED_DATE	COUNT	PRICE
1	C.Dickens	A Tale of Two Cities	1986-08-09	12	30.0
2	C.Dickens	Great Expectations	1861-02-03	18	32.0

Row count: 25

You can perform the above search in the Analytics REST API using the following request. For more information, see [Drilling Down Through Categories via REST API](#).

```
POST https://localhost:9443/analytics/drillDown

{
    "tableName": "BOOK_STORE",
    "categories": [
        {
            "fieldName": "AUTHOR",
            "path": [ "C.Dickens" ]
        }
    ],
    "query": "timestamp : [1243214324532 TO 4654365223]",
    "recordStart": 0,
    "recordCount": 100
}
```

Extracting sub categories

Another use of facets is to extract the sub categories of a category. You can retrieve the immediate sub-categories of a given category (which are represented in a JSON array), using the relevant API. The API returns the immediate subcategories of the given category in the corresponding table.

For an example, in a record which represents a book, you can define the PUBLISHED_DATE field as a facet when you are persisting the event stream as shown below.

The screenshot shows the 'Define New Event Stream' page. In the 'Payload Data Attributes' table, the 'PUBLISHED_DATE' row is selected, and its 'Attribute Type' is highlighted with a red box. The other rows show attributes like TITLE, AUTHOR, COUNT, and PRICE with their respective attribute types: STRING, STRING, INTEGER, and FLOAT.

Click the **Simulate** option of the event stream to simulate sending events to the created even stream.

Home > Manage > Event > Streams

Available Event Streams

[+ Add Event Stream](#)

2 Event streams available

Event Stream Id	Event Stream Description	Actions
BOOK_STORE:1.0.0		
org.wso2.sample.httpd.logs:1.0.0	Sample of Httpd logs	

Send its values as a JSON string array for the attribute which you defined as a facet as shown in the example below.

Home > Manage > Event > Streams > List

Event Stream Simulator

Send single event

Event Stream Name *	BOOK_STORE:1.0.0
Stream Attributes	
Payload Attributes	
TITLE(string) *	The Sun Also Rises
AUTHOR(string) *	E.Hemingway
PUBLISHED_DATE(string) *	[1926,'08,'09]
COUNT(int) *	42
PRICE(float) *	45
<input type="button" value="Send"/> <input type="button" value="Clear"/>	

e.g., If the above BOOK_STORE table contains the below four records with the corresponding values for PUBLISHED_DATE attribute, the REST API returns the sub categories of ['1926'], which are '08', '04', and the sub categories of ['1926' , '08'], which are '09' and '10'.

- Record 1 - **PUBLISHED_DATE**: ['1926' , '08' , '09']
- Record 2 - **PUBLISHED_DATE**: ['1926' , '04' , '02']
- Record 3 - **PUBLISHED_DATE**: ['1816' , '09' , '01']
- Record 4 - **PUBLISHED_DATE**: ['1926' , '08' , '10']

You can retrieve PUBLISHED_DATE as a specific category and its sub categories using the [Data Explorer](#) as shown below.

Home > Manage > Interactive Analytics > Data Explorer

Data Explorer

Search

Table Name* **BOOK_STORE**

Search By Date Range By Query

PUBLISHD_DATE

PUBLISHD_DATE Select a category

Home > Manage > Interactive Analytics > Data Explorer

Data Explorer

Search

Table Name* **BOOK_STORE**

Search By Date Range By Query

PUBLISHD_DATE

PUBLISHD_DATE Select a category

You can perform the above search in the Analytics REST API using the following requests. For more information, see [Drilling Down Through Categories via REST API](#).

```
POST https://localhost:9443/analytics/facets
{
    "tableName" : "BOOK_STORE",
    "fieldName" : "PUBLISHED_DATE",
    "categoryPath" : ["1926"],
    "query" : "timestamp : [1213343534535 TO 465464564644]"
}
```

```
POST https://localhost:9443/analytics/facets
{
    "tableName" : "BOOK_STORE",
    "fieldName" : "PUBLISHED_DATE",
    "categoryPath" : ["1926", "08"],
    "query" : "timestamp : [1213343534535 TO 465464564644]"
}
```

Performing a drill down search

This denotes a hierarchical implementation of a collection of several categories of attributes within one attribute. The values of a set of records, which you can use to classify the records can be indexed as facets. Those fields which are indexed as facets are used to implement faceted search and drill-down.

For an example, in a record which represents a book, you can define the `PUBLISHED_DATE` field as a facet when you are persisting the event stream as shown below.

The screenshot shows the 'Define New Event Stream' page. Under 'Enter Event Index Details', the 'Persist Event Stream' checkbox is checked. The 'Record Store' dropdown is set to 'EVENT_STORE'. In the 'Payload Data Attributes' section, there is a table with columns: Persist Attribute, Attribute Name, Attribute Type, Primary Key, Index Column, and Score Param. The 'PUBLISHED_DATE' row has its 'Attribute Type' set to 'FACET' and is highlighted with a red border. Other attributes listed are TITLE, AUTHOR, COUNT, and PRICE, each with their respective types (STRING, STRING, INTEGER, FLOAT). Below this table, under 'Arbitrary Data Attributes', it says 'No arbitrary data attributes are defined'. At the bottom, there are buttons for 'Back' and 'Save Event Stream'.

Click the **Simulate** option of the event stream to simulate sending events to the created even stream.

The screenshot shows the 'Available Event Streams' page. It lists two event streams: 'BOOK_STORE:1.0.0' and 'org.wso2.sample.httpd.logs:1.0.0'. For each stream, there is a 'Actions' column containing three buttons: 'Simulate' (highlighted with a red border), 'Delete', and 'Edit'. The 'Simulate' button is specifically highlighted in the first row for the 'BOOK_STORE' stream.

For the attribute which you defined as a facet, you need to send its values as a JSON string array as shown in the example below .

Home > Manage > Event > Streams > List

Event Stream Simulator

Send single event

The screenshot shows a web-based form for sending a single event to an event stream. At the top, there is a dropdown menu labeled "Event Stream Name *" with the value "BOOK_STORE:1.0.0". Below this is a section titled "Stream Attributes" which contains a "Payload Attributes" table.

Payload Attribute	Value
TITLE(string) *	The Sun Also Rises
AUTHOR(string) *	E.Hemingway
PUBLISHED_DATE(string) *	['1926','08','09']
COUNT(int) *	42
PRICE(float) *	45

At the bottom of the form are two buttons: "Send" and "Clear".

You can use a facet to filter the books by the published date as a specific category and published year/month/date as its sub categories using the [Data Explorer](#) as shown below.

Home > Manage > Interactive Analytics > Data Explorer

[Help](#)

Data Explorer

The screenshot shows the "Data Explorer" interface. In the "Search" section, the "Table Name" is set to "BOOK_STORE" and the "Facet" is set to "PUBLISHED_DATE". The search criteria are set to "By Date Range". The date range is specified as "1866" for the year, "05" for the month, and "03" for the day. A "Select a category" dropdown is also present. At the bottom of the search section are "Search" and "Reset" buttons.

In the "Results" section, the results are displayed in a table for the "BOOK_STORE" table. The columns are TITLE, AUTHOR, PUBLISHED_DATE, COUNT, PRICE, and _timestamp. One record is shown: "A Tale of Two Cities" by C.Dickens, published on 1866-05-03, with a count of 12, price of 20.0, and timestamp of 2015-08-19 18:32:44 IST. The "Row count" is listed as 25.

You can perform the above search in the Analytics REST API using the following request. It will return the records which match the drill down search. For more information, see [Retrieving Specific Records through a Drill Down Search via REST API](#).

```
POST https://localhost:9443/analytics/drillDown

{
    "tableName": "BOOK_STORE",
    "categories": [
        {
            "fieldName": "PUBLISHED_DATE",
            "path" : [ "1866", "05", "03" ]
        }
    ],
    "query" : "timestamp : [1243214324532 TO 4654365223]",
    "recordStart" : 0,
    "recordCount" : 100
}
```

⚠ In the above example, PUBLISHED_DATE is a facet of which values are defined in a three element JSON array. In this example, "05" is a sub-category of "1866", and "03" is a sub-category of "05". This information is useful to perform drill down search operations. If you want to retrieve records of which the PUBLISHED_DATE starts with "1866", provide only "1866" in a JSON array as the value of the facet in the REST API request. Similarly, if you want to retrieve records of which the PUBLISHED_DATE is "1866/05/ANY_DAY", provide ["1866", "05"] as the value of the facet in the REST API request.

Also you can perform the above search in the Analytics REST API using the following request. It will return the number of records which match the drill down search. For more information, see [Retrieving the Number of Records Matching the Drill Down Criteria via REST API](#).

```
POST https://localhost:9443/analytics/drillDownCount

{
    "tableName": "BOOK_STORE",
    "categories": [
        {
            "fieldName": "PUBLISHED_DATE",
            "path" : [ "1866", "05", "03" ]
        }
    ],
    "query" : "timestamp : [1243214324532 TO 4654365223]"
    "recordStart" : 0,
    "recordCount" : 100
}
```

Searching data within a value range

This denotes using facets to filter data based on a value range of an attribute which is defined as a facet.

For an example, in a record which represents a book, you can define the PRICE field as a facet when you are persisting the event stream as shown below.

✓ Define the field based on which you want to search data as numeric (INTEGER, FLOAT etc.), and as an **Index Column**.

Define New Event Stream

Enter Event Index Details

Persist Event Stream

Record Store
EVENT_STORE

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
<input checked="" type="checkbox"/>	TITLE	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	AUTHOR	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	PUBLISHED_DATE	STRING	<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 checked="" type="checkbox"/>	COUNT	INTEGER	<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 :

Click the **Simulate** option of the event stream to simulate sending events to the created even stream.

Available Event Streams

 [Add Event Stream](#)

2 Event streams available

Event Stream Id	Event Stream Description	Actions
BOOK_STORE:1.0.0		
org.wso2.sample.httpd.logs:1.0.0	Sample of Httpd logs	

You can perform the above search in the Analytics REST API using the following request. For more information, see [Retrieving the Event Count of Range Facets](#).

in this WSO2 DAS version, the Data Explorer does not support performing search operations on range facets.

```

POST https://localhost:9443/analytics/rangecount

{
    "tableName": "BOOK_STORE",
    "rangeField" : "PRICE",
    "ranges" : [
        {
            "label" : "20USD - 30USD",
            "from" : 20,
            "to" : 30
        },
        {
            "label" : "30USD - 40USD",
            "from" : 30,
            "to" : 40
        }
    ],
    "query" : "*:*"
}

```

Score parameters

Score parameters are used as function parameters of score functions. You can define only INTEGER, DOUBLE, FLOAT or LONG type fields as score parameters. You can define score parameters when you [persist event stream definitions](#) along with indices. Score parameters are used in the implementations of the following REST APIs.

- Drilling Down Through Categories via REST API
- Retrieving the Number of Records Matching the Drill Down Criteria via REST API

Score functions

Score functions are used to override the default score of a record which has facet fields. Default score is 1. The default score is used to retrieve the [drill down record count](#) and [sub categories of a category](#). If you override the default value, then the score of that record will be the evaluation of the score function.

Retrieving the record score matching a drill down search

For an example, in a record which represents a book you can define PRICE and DISCOUNT as score parameters that can be used for the following score function example: 'price - discount'. You need to define these fields as score parameters and index columns when persisting the event stream as shown below.

Edit Event Stream

Enter Event Stream Details

Persist Event Stream

Record Store
EVENT_STORE

Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
<input checked="" type="checkbox"/>	TITLE	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	AUTHOR	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	PUBLISHED_DATE	FACET	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	COUNT	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	PRICE	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	DISCOUNT	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Arbitrary Data Attributes
No arbitrary data attributes are defined

Attribute Name : Attribute Type : Primary Key : Index Column : Score Param :

Click the **Simulate** option of the event stream to simulate sending events to the created even stream.

Available Event Streams

2 Event streams available

Event Stream Id	Event Stream Description	Actions
BOOK_STORE:1.0.0		<input type="button" value="Simulate"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
org.wso2.sample.httpd.logs:1.0.0	Sample of Httpd logs	<input type="button" value="Simulate"/>

For an example, consider an event stream with the following two records.

```
record1 (Book1) :
TITLE : Oliver Twist,
AUTHOR : C.Dickens,
PUBLISHED_DATE :["1866", "08", "03"],
COUNT : 22,
PRICE : 30.00,
DISCOUNT : 10.00

record2 (Book2) :
TITLE : Great Expectations,
AUTHOR : C.Dickens,
PUBLISHED_DATE : ["1826", "09", "14"],
COUNT : 22,
PRICE : 50.00,
DISCOUNT : 12.00
```

Score parameters are useful when you want to use the [drill down count API](#) to get the sum of the scores of records. If you invoke the API to retrieve the drilldown record count without a score function, then the score of each record is 1 (i.e. the default value). Therefore, the API returns the number of records. You can define a score function as “price - discount” as shown in the below REST API request. For more information, see [Drilling Down Through Categories via REST API](#).

```
POST https://localhost:9443/analytics/drillDownCount

{
    "tableName": "BOOKS_STORE",
    "categories": [
        {
            "fieldName": "AUTHOR",
            "path" : [ "C.Dickens" ]
        }
    ],
    "scoreFunction" : "PRICE - DISCOUNT"
}
```

Now, the score of each record is the output of the score function. Therefore, the API returns 58 as the sum of the effective prices after applying the discount.

Retrieving the record score based on specific categories

You can use the REST API to retrieve the score of each record (after applying the score function) based on specific categories as shown below. For more information, see [Retrieving the Number of Records Matching the Drill Down Criteria via REST API](#).

```
POST https://localhost:9443/analytics/facets
{
    "tableName" : "BOOK_STORE",
    "fieldName" : "PUBLISHED_DATE",
    "categoryPath" : ["1866", "08"],
    "query" : "timestamp : [1213343534535 TO 465464564644]",
    "scoreFunction" : "PRICE-DISCOUNT"
}
```

The sample out put of the above request is as follows. It denotes the following.

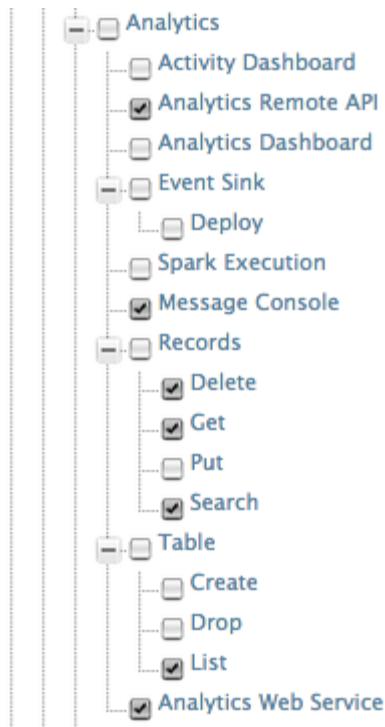
- Output of the score function of the records with the PUBLISHED_DATE as ["1866", "08", "23"] is 15.
- Output of the score function of the records with the PUBLISHED_DATE as ["1866", "08", "12"] is 25.

```
{
    "categoryPath" : ["1866", "08"],
    "categories" : {"23" : 15, "12" : 25}
}
```

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 message console.

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.

Home > Manage > Interactive Analytics > Data Explorer

Data Explorer

Search

Table Name* **BOOK_STORE**

Maximum Result Count **1000**

Search By Date Range By Primary Key By Query

Results

Note: Total record count for the table is not available.

BOOK_STORE							
	ID	Title	Author	Published-Date	Category	Count	_timestamp
1	00001	A Tale of Two Cities	Charles Dickens	'1859','04','01'	Fiction,Classic,	120	2015-10-28 10:00:31 IST
2	00002	The Casual Vacancy	J.K.Rowling	2012,09,27	Fiction,Modern,	120	2015-10-28 10:02:10 IST
3	00003	Cocktail Time	P.G. Wodehouse	1958,06,12	Fiction,Humour,	120	2015-10-28 10:03:54 IST
4	00005	The Victorians	A.N. Wilson	1988	History,Victorian Era	200	2015-10-28 10:07:22 IST
5	00006	Elizabeth the Queen	Alison Weir	1998	History,Elizabethan Era	200	2015-10-28 10:09:03 IST
6	00007	How to Win Friends and Influence People	Dale Carnegie	1936	Self-Help,Sociology	200	2015-10-28 10:12:54 IST
7	00008	The Wealth of Nations	Adam Smith	1776	Economics,Classical	200	2015-10-28 10:14:51 IST

<< < 1 2 ... 99 100 > >> Go to page: **1** Row count: **10**

Showing 1-10 of 1000

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.

i 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.

Home > Manage > Interactive Analytics > Data Explorer

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
1	00001	A Tale of Two Cities	Charles Dickens	'1859','04','01'	Fiction,Classic,	120	2015-10-28 10:00:31 IST
2	00002	The Casual Vacancy	J.K.Rowling	2012,09,27	Fiction,Modern,	120	2015-10-28 10:02:10 IST
3	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.

The screenshot shows the Data Explorer interface with the following details:

- Search Section:**
 - Table Name: BOOK_STORE
 - Maximum Result Count: 1000
 - Search Type: By Primary Key
 - Author: Charles Dickens
 - Title: A Tale of Two Cities
- Results Section:**
 - Total Records: 1
 - BOOK_STORE Table Data:

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.

The screenshot shows the Data Explorer interface with the following details:

- Search Section:**
 - Table Name: BOOK_STORE
 - Maximum Result Count: 1000
 - Search Type: By Query
 - Query: Title:"The Casual Vacancy"
- Results Section:**
 - Total Records: 1 (135 ms)
 - BOOK_STORE Table Data:

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

 - Row count: 10

Searching by facets

Follow the steps below to search for records of an existing table using facets.

i For more information on facets, see [Searching Data By Categories](#).

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**.

- Click **Search**. You view the filtered records of the selected table as shown below.

The screenshot shows the Data Explorer interface for the 'BOOK_STORE' table. The search query is set to 'Published-Date' with values 1958, 06, 12. The results table shows one record: ID 00003, Title 'Cocktail Time', Author 'P.G. Wodehouse', Published-Date '1958-06-12', Category 'Fiction,Humour', Count 120, and timestamp '2015-10-28 10:03:54 IST'. A red box highlights the search facet dropdown and its values.

You can also perform searches with facets using combined Lucene queries in the message console.

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 it results by providing an valid Lucene query.

Let's consider an example, a transaction is being processed and going through several subsystems and if you want to find out the status of the particular transaction such as whether it has completed or at which sub system currently it's being processed, etc can be filtered by sending the events to our WSO2 DAS with same activity ID, and then you can search through the events collected from different sub systems.

The activity monitoring dashboard groups all events which belongs to same activity id, and provide you the list of activity ids, and then you can drill down the activity id to see what are the actual events that has fired and then take your decisions based on the analysis.

Below sample use case takes you over the capabilities of 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

If you want to use activity dashboard to search your events, then you need to define the activityId field as the 'FACET' field and enable the indexing for that field as given below.



Also you can search the activity by field, and if you want to search by any particular fields other than timestamp, then you also need to index them appropriately. In below provided example the user is required to search by `meta_host`, `meta_http_method`, `meta_message_type`, and `operation_name`. Therefore the user have to enable indexes to those fields.

Edit Event Stream

Enter Event Stream Details

Persist	Column Name	Type	Primary Key	Index	Score Param
<input checked="" type="checkbox"/>	meta_character_set_encoding	STRING	<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 checked="" type="checkbox"/>	meta_http_method	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	meta_message_type	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	meta_remote_address	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	meta_remote_host	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	meta_service_prefix	STRING	<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 checked="" type="checkbox"/>	meta_transport_in_url	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	correlation_activity_id	FACET	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	SOAPBody	STRING	<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 checked="" type="checkbox"/>	message_direction	STRING	<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 checked="" type="checkbox"/>	operation_name	STRING	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	service_name	STRING	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	timestamp	LONG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

[Back](#) [Edit Event Stream](#)

Publish events with the activity ID

As we have mentioned above, the event needs to have mandatory 'activityId' field in the correlation data of the event and the field needs to be as JSON string to use it as FACET type. Therefore convert your activityId as JSON string and publish to WSO2 DAS. For example, sample JSON formed activity shown below.

```
[ 1ceccb16-6b89-46f3-bd2f-fd9f7ac447b6 ]
```

Using the activity monitoring dashboard

If you want to perform the search within any time period, then select the data and time for 'From time' and 'To time' respectively. If you don't select any time duration for the field, then it will search through the full time range. But it's advisable to use this time range, if you have events collected and hence searching through the entire list of events will have some processing/memory needs.

Home > Dashboard > Activity Dashboard

[Activity Search](#)

Time Search

From Time:

To Time:

[Advanced Search](#)

Search Query:

May 2015

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: The time from which activities you are interested on.

To Time: The time to which activities you are interested on.

Advanced Search

Search Query:

May 2015

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: 13
Minute: 00

served.

You also can give any Lucene queries to further filter the results from the time range you have given above. You can add any number of nested queries, which spans over any number of tables. Below given is the sample query that you can search.

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 --> A1[ORG_WS02_BAM_ACTIVITY_MONITORING : meta_remote_host:"localhost" AND meta_http_method :"POST"]
    AND --> A2[ORG_WS02_DAS_ACTIVITY_MONITORING : meta_remote_host:"localhost" AND meta_http_method :"POST"]
    A2 --> OR
  
```

You will get the list of activity ids as below, and when you click on each you will see a first 10 records for the activities. If you want, you can click on view more.

Activities Search Result

[\[3ceccb16-6b89-46f3-bd2fffd9f7ac447b6\]](#)

ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793621.0787071459800018E-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..](#)[\[2ceccb16-6b89-46f3-bd2fffd9f7ac447b6\]](#)[\[5ceccb16-6b89-46f3-bd2fffd9f7ac447b6\]](#)[\[1ceccb16-6b89-46f3-bd2fffd9f7ac447b6\]](#)[\[4ceccb16-6b89-46f3-bd2fffd9f7ac447b6\]](#)

1 - 5 of 5

Then you can click on any record that you like to view to see the full record content.

Record: ORG_WSO2_BAM_ACTIVITY_MONITORING : 14325562793630.4120781945270447

meta_http_method	POST
correlation_activity_id	[3ceccb16-6b89-46f3-bd2fffd9f7ac447b6]
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_Proxy?urn:uuid:c70bae36-b163-4f3e-a341-d7079c58f1ba&n: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 Data By Categories](#), 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" A N D category:"child_fiction"</code>
Search for records that match one of the matching criteria when multiple matching criteria is provided	<code><ATTRIBUTE_NAME>:<ATTRIBUTE_VALUE> OR <ATTRIBUTE_NAME>:<ATTRIBUTE_VALUE></code>	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 author and subject 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	<code><ATTRIBUTE_NAME>:[<MINIMUM_VALUE> TO <MAXIMUM_VALUE>]</code>	If you are searching for a book for which the count is between 100 and 200, you can use 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.

Spark SQL

Spark
Streaming

MLlib

GraphX

Spark Core Engine

i 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](#)
- [Spark Troubleshooting](#)

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
SparksSQL> |
```

Previous 1 Next

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.

[Help](#)

New Analytics Script Information

Script Name *

MyFirstAnalyticsScript

The name of the actual analytics script.

Spark SQL Queries *

```

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;

```

Position: Ln 6, Ch 26 | Total: Ln 6, Ch 289 | Toggle editor

Schedule Script

Cron Expression

0 0/5 * * * ?

The cron expression for the rate of analytics script to be executed

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

0 * * * *

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,..)

4. Click **Execute**, to execute the provided queries. This will display the results as follows.

The screenshot shows the 'Schedule Script' dialog. In the 'Cron Expression' field, '0 * * * ?' is entered. Below it, the 'Execution Results' section displays the output of the script execution:

```

9 |SELECT substr(summary, 1, 5) FROM Log;
10 |
11 |SELECT LAST(ip) FROM Log;
12 |
13 |

Position: Ln 13, Ch 5 Total: Ln 13, Ch 296
 Toggle editor

```

Execution 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
192.168.0.11

5. Click **Add**, to add the configured script.

Editing a script

Follow the steps below to edit an existing Analytics 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 **Edit** for the script you want to edit.

Home > Manage > Batch Analytics > Scripts [Help](#)

Available Analytics Scripts

Scripts	Actions			
MyFirstAnalyticsScript				
SparkScript				

[+ Add New Analytics Script](#)

4. 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.

Help

Edit Analytics Script – MyFirstAnalyticsScript

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;
```

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	
SparkScript	

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 [Help](#)

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
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](#)

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 or other.

- Creating the table using Carbon Analytics as the provider
- Creating the table using Carbon JDBC as the provider
- Creating the table using other relation providers

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 exists), using data from Carbon analytics. Carbon analytics refer to either the built-in H2 database or any external database which 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 optional.</p> <p>⚠ You do not need to specify a schema for a table which already exists in the DAL, as its schema would be inferred. Specifying a schema again for an existing table with the given name will overwrite the initial schema.</p> <p>i 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. All indexed columns should be of numeric type. • <code>-sp</code> denotes an indexed column with score param. 	schema "house_id INT, household_id INT, plug_id INT, usage FLOAT -sp, composite FACET -i"
primaryKeys	Primary key of the table in the DAL. This is optional. Assign primary keys if and only if you have provided a schema.	primaryKeys "household_id, plug_id"
recordStore	The Analytics Record Store in which this table is created.	recordStore "EVENT_STORE" <p>i The default Analytics Record Store used by CarbonAnalytics is the PROCESSED_DATA_STORE.</p>

Creating the table using Carbon JDBC as the provider

Use the following query syntax to create a table in the Spark environment using data from Carbon JDBC.

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

Options in the above syntax are described below.

Option	Description	Example
<code>dataSource "<datasource name>"</code>	The name of the data source from which data should be obtained for the temporary table.	<code>dataSource "test"</code>

tableName <table name>	The name of the table in the selected data source from which data should be obtained for the temporary table.	tableName "TEST.PEOPLE"
------------------------	---	----------------------------

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.

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.

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

Publishing Events Using Apache Spark

You can publish events from WSO2 DAS using Spark SQL queries. This feature can be used to alert any interested parties whenever the content of an existing Spark table gets changed. For instance, a scheduled Spark script can periodically check on a Spark table for specific conditions to be met. If the conditions are satisfied, an event can be published downstream to notify the interested parties. There are three functions involved in publishing events from WSO2 DAS using Spark as follows.

- [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 (receiverURL "<das_receiver_url>",
         authURL "<das_receiver_auth_url>",
         username "<user_name>",
         password "<password>",
         streamName "<stream_name>",
         version "<stream_version>",
         description "<description>",
         nickName "<nick_name>",
         payload "<payload>"
       );
INSERT OVERWRITE TABLE <table_name> <select_query>;
```

The parameters of the above query are described below.

Parameter	Description
-----------	-------------

<table_name>	Name of the Spark table which maps to the created event stream.
<das_receiver_url>	The DAS Thrift receiver URL set. (e.g., tcp://10.100.0.40:7611)
<das_receiver_auth_url>	The DAS Thrift receiver auth URL set. (This is an optional parameter. If no value is specified, the default auth URL set is derived from receiver URL set, e.g ., ssl://10.100.0.40:7711).
<user_name>	Username for connecting DAS receiver node.
<password>	Password for connecting DAS receiver node.
<stream_name>	Name of the stream which will be published
<stream_version>	Version of the stream (E.g. 1.0.0)
<description>	Description about the stream being created (This is optional.)
<nick_name>	A friendly display name for the stream being created (This is optional.)
<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 to be published to the event stream from the existing Spark table. (E.g. select ip_address, name, message from EventStream)

Now, 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 instructions on configuring 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(“cusom”, “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;
    }
}

```

i 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)
{
}

```

i Method overloading for UDFs is not supported. Different UDFs should have different method names for the expected behaviour.

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-configuration.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.

i Spark adds all the methods in the specified UDF class as custom UDFs.

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

Realtime Analytics Using Siddhi

Following sections describe how you can perform realtime analytics using Siddhi in WSO2 DAS.

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

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 :

Export Stream Value Of : Streamid :

```

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.

i 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.

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

- i** 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. 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.

The screenshot shows the 'Create a New Execution Plan' page. In the 'Query Expressions' section, there are two input fields: 'Import Stream*' and 'As :'. Below these are 'Export Stream' and 'Value Of :'. A large text area contains Siddhi Query Language code:

```

1 /* Enter a unique ExecutionPlan */
2 @Plan:name('NewExecutionPlan')
3
4 /* Enter a unique description for ExecutionPlan */
5 @Plan:description('This is the description for my NewExecutionPlan')
6
7 /* define streams/tables and write queries here ... */
8
9 @Import('inStream:1.0.0')
10 define stream inStream (meta_temperature double, meta_roomNumber int);
11
12 @Export('outStream:1.0.0')
13 define stream outStream (meta_temperature double, meta_roomNumber int);
14
15 from inStream[meta_temperature > 75]
16 select meta_temperature, meta_roomNumber
17 insert into outStream
18

```

At the bottom of the text area are 'Validate Query Expressions' and 'Add Execution Plan' buttons.

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.



- 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.

Editing a deployed execution plan

Follow the procedure below to edit 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 **Edit** in the row of the execution plan you want to edit. The **Edit Execution Plan Configuration** page will open.
3. Edit the execution plan as required and click **Update**.

i 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/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
<code>@dist (parallel='<number of Storm tasks>')</code>	The number of storm tasks in which the query should be run parallel.	<code>@dist(parallel='4')</code>

@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;
```



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
 - In
 - Join
- Event Trigger
 - Event Trigger Definition
- 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.

Term	Definition
Event Stream	Logical series of events ordered in time.
Event Stream Definition	Defines event streams . An event stream has a unique name and a set of attributes assigned specific types, with uniquely identifiable names defining its schema.
Event	An event is associated with only one event stream, and all events of that stream have an identical set of attributes assigned specific types (or the same schema). Event will contain timestamp and the attribute values according to the schema.
Attribute	An attribute has a unique name within the event stream. The attribute type can be <code>string</code> , <code>int</code> , <code>long</code> , <code>float</code> , <code>double</code> , <code>bool</code> or <code>object</code> .
Query	A logical construct that derives new streams by combining existing streams. A query contains one or more input streams, handlers to modify those input streams, and an output stream to which it publishes its output events.
Partition	A logical container that processes a subset of the queries based on a pre-defined rule of separation.
Event Table	A structured representation of stored data, allowing stored data to be accessed and manipulated at runtime.

Siddhi have 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 <code>roomNo</code> and <code>temp</code> attributes from the <code>TempStream</code> 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 <code>select</code> statement.</p> <p>e.g., Use one of the following queries to select all the attributes in the <code>TempStream</code> stream.</p> <pre data-bbox="465 369 856 454">from TempStream select * insert into NewTempStream;</pre> <p>or</p> <pre data-bbox="481 601 856 665">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 <code>roomNo</code> to <code>roomNumber</code> and <code>temp</code> to <code>temperature</code>.</p> <pre data-bbox="481 918 1175 1024">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 <code>as</code>.</p> <p>e.g.,</p> <pre data-bbox="481 1235 954 1341">from TempStream select roomNo, temp, 'C' as scale insert into RoomTempStream;</pre>
Using mathematical and logical expressions	<p>This involves using attributes with mathematical and logical expressions to the precedence order given below, and assigning them to the output attribute using <code>as</code>.</p> <p>Operator precedence</p>

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, substraction	temp * 9/5 + 32
< <=	Comparisons: less-than, 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 the DAS, see [Datasources](#) in the Admin Guide.

E.g. Create an event table with name RoomTypeTable having attributes room number as int & type as string, backed by MySQL table named RoomTable from the database cepdb located at localhost:3306 having user name "root" and password "root".

```
@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.

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 oder 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 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 Extensions

Siddhi supports an extension architecture to support custom code and functions to be incorporated with Siddhi in a seamless manner. Extension will follow the 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 interfaces. 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

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|f
loat|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 everted 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`

`time`

```
<event> time (<int|long|time> windowTime)
```

- **Extension Type:** Window
- **Description:** Sliding time window, that holds events for that arrived during last `windowTime` period, and gets updated on every event arrival and expiry.
- **Parameter:** `windowTime`: The sliding time period for which the window should hold events.
- **Return Type:** Return current and expired events.
- **Examples:** `time(20)` for processing events arrived in last 20 milliseconds. `time(2 min)` for processing events arrived in last 2 minutes.

`timeBatch`

```
<event> timeBatch (<int|long|time> windowTime)
```

- **Extension Type:** Window
- **Description:** Batch (tumbling) time window, that holds events arrived between `windowTime` periods, and gets updated for every `windowTime`.
- **Parameter:** `windowTime`: The batch time period for which the window should hold events.
- **Return Type:** Return current and expired events.
- **Examples:** `timeBatch(20)` for processing events arrived every 20 milliseconds. `time(2 min)` for processing events arrived every 2 minutes.

`length`

```
<event> length (<int> windowLength)
```

- **Extension Type:** Window
- **Description:** Sliding length window, that holds last windowLength events, and gets updated on every event arrival and expiry.
- **Parameter:** **windowLength** : The number of events that need to be in a sliding length window.
- **Return Type:** Return current and expired events.
- **Examples:** `length(10)` for processing last 10 events. `time(200)` for processing last 200 events.

lengthBatch

```
<event> lengthBatch (<int> windowLength)
```

- **Extension Type:** Window
- **Description:** Batch (tumbling) length window, that holds up to windowLength events, and gets updated on every windowLength event arrival.
- **Parameter:** **windowLength** : For the number of events the window should tumble.
- **Return Type:** Return current and expired events.
- **Examples:** `lengthBatch(10)` for processing 10 events as a batch. `time(200)` for processing 200 events as a batch.

externalTime

```
<event> time (<long> timestamp, <int|long|time> windowTime)
```

- **Extension Type:** Window
- **Description:** Sliding time window based on external time, that holds events for that arrived during last windowTime period from the external timestamp, and gets updated on every monotonically increasing timestamp.
- **Parameter:** **timestamp** : The time which the window determines as current time and will act upon, the value of this parameter should be monotonically increasing.
- **Parameter:** **windowTime**: The sliding time period for which the window should hold events.
- **Return Type:** Return current and expired events.
- **Examples:** `externalTime(eventTime,20)` for processing events arrived in last 20 milliseconds from the eventTime. `externalTime(eventTimestamp, 2 min)` for processing events arrived in last 2 minutes from eventTimestamp.

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

Siddhi supports custom codes within queries. You can implement Windows, Transformers, OutputAttributeProcessors, Functions (Conditions and Expressions) in a pluggable manner using the current implementation.



Note

These are subjected to be changed in a future release.

Use of Namespaces and Functions in Siddhi Extensions

Siddhi allows extensions to have namespaces and function names, enabling users to easily identify the behaviour of the extensions when writing queries.

To add namespace and function names to the extension, use the following Java annotation in the extension class.

```
@SiddhiExtension(namespace = "testExt", function = "unique")
public class UniqueWindowProcessor extends WindowProcessor {
    ...
}
```

You can refer to the above class in the query as follows:

```
from StockExchangeStream[price >= 20]#window.testExt:unique(symbol)
select symbol, price
insert into StockQuote
```

The following sections explains how we can 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 Window Extension

To write a custom Window, create a class extending "**org.wso2.siddhi.core.query.processor.stream.window.WindowProcessor**" and create an appropriate .siddhiext extension mapping file, compile the class, and build the jar containing the .class and .siddhiext files. Add them to the Siddhi class path. In the case of running them on WSO2 DAS add the jar to <DAS_HOME>/repository/components/lib.

For example, Window extension created with namespace "custom" and function name "lastUnique" can be referred in the query as follows:

```
from StockExchangeStream[price >= 20]#window.custom:lastUnique(symbol,5)
select symbol, price
insert into StockQuote;
```

For the Window extension to be used in a Join Query the Window Extension should be findable. To make the Window findable it should implement the "**org.wso2.siddhi.core.query.processor.stream.window.FindableProcessor**" Interface.

E.g. Implementation can be found below;

```
/*
 * Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
 *
 * Licensed 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.core.query.processor.stream.window;

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.ExpressionExecutor;
import org.wso2.siddhi.core.executor.VariableExpressionExecutor;
import org.wso2.siddhi.core.query.processor.Processor;
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.expression.Expression;

import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class LastUniqueWindowProcessor extends WindowProcessor implements
FindableProcessor{
    private ConcurrentHashMap<String, StreamEvent> map = new ConcurrentHashMap<String,
StreamEvent>();
    private VariableExpressionExecutor[] variableExpressionExecutors;

    /**
     * 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) {
        variableExpressionExecutors = new
VariableExpressionExecutor[attributeExpressionExecutors.length];
        for (int i = 0; i < attributeExpressionExecutors.length; i++) {
            variableExpressionExecutors[i] =(VariableExpressionExecutor)
attributeExpressionExecutors[i];
        }
    }

    /**
     * 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) {
    ComplexEventChunk<StreamEvent> complexEventChunk = new
ComplexEventChunk<StreamEvent>();

    StreamEvent streamEvent = streamEventChunk.getFirst();
    while (streamEvent != null) {
        StreamEvent clonedEvent = streamEventCloner.copyStreamEvent(streamEvent);
        clonedEvent.setType(StreamEvent.Type.EXPIRED);

        StreamEvent oldEvent = map.put(generateKey(clonedEvent), clonedEvent);
        if (oldEvent != null) {
            complexEventChunk.add(oldEvent);
        }
        StreamEvent next = streamEvent.getNext();
        streamEvent.setNext(null);
        complexEventChunk.add(streamEvent);
        streamEvent = next;
    }
    nextProcessor.process(complexEventChunk);
}

/**
 * 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() {
    //Do nothing
}

/**
 * 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() {
    //Do nothing
}

/**
 * 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[]{map};
}

/**
 * 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) {
    map = (ConcurrentHashMap<String, StreamEvent>) state[0];
}

/**
 * To find events from the processor event pool, that 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, map.values(), 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);  
  
}  
  
private String generateKey(StreamEvent event) {  
    StringBuilder stringBuilder = new StringBuilder();  
    for (VariableExpressionExecutor executor : variableExpressionExecutors) {  
        stringBuilder.append(event.getAttribute(executor.getPosition()));  
    }  
    return stringBuilder.toString();  
}
```

```
}
```

Writing a Custom Aggregate Function

To implement a custom Aggregate Function, create a class extending "**org.wso2.siddhi.core.query.selector.attribute.aggregator.AttributeAggregator**" and create an appropriate .siddhiext extension mapping file, compile the class, and build the jar containing the .class and .siddhiext files. Add them to the Siddhi class path. In the case of running them on WSO2 DAS add the jar to <DAS_HOME>/repository/components/lib.

For example, Aggregate Function extension created with namespace "custom" and function name "std" can be referred in the query as follows:

```
from StockExchangeStream[price >= 20]
select symbol, custom:std(price) as stdPrice
insert into StockQuote;
```

E.g. Implementation can be found below;

```
/*
 * Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
 *
 * Licensed 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.core.query.selector.attribute.aggregator;

import org.wso2.siddhi.core.config.ExecutionPlanContext;
import org.wso2.siddhi.core.exception.OperationNotSupportedException;
import org.wso2.siddhi.core.executor.ExpressionExecutor;
import org.wso2.siddhi.query.api.definition.Attribute;

import java.util.Arrays;

public class StrandedDeviationAggregateFunction extends AttributeAggregator {

    private final Attribute.Type type = Attribute.Type.DOUBLE;
    private double mean, oldMean, stdDeviation, sum;
    private int count = 0;

    /**
     * The initialisation method for FunctionExecutor
     *
     * @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) {
    if (attributeExpressionExecutors.length != 1) {
        throw new OperationNotSupportedException("Stddev aggregator has to have
exactly 1 parameter, currently " +
                attributeExpressionExecutors.length + " parameters provided");
    }
}

@Override
public Attribute.Type getReturnType() { return type; }

@Override
public Object processAdd(Object data) {
    count++;
    double value = (Double) data;

    if (count == 1) {
        sum = mean = oldMean = value;
        stdDeviation = 0.0;
    } else {
        oldMean = mean;
        sum += value;
        mean = sum / count;
        stdDeviation += (value - oldMean)*(value - mean);
    }

    if (count < 2) {
        return 0.0;
    }
    return Math.sqrt(stdDeviation / count);
}

@Override
public Object processRemove(Object data) {
    count--;
    double value = (Double) data;

    if (count == 0) {
        sum = mean = 0;
        stdDeviation = 0;
    } else {
        oldMean = mean;
        sum -= value;
        mean = sum / count;
        stdDeviation -= (value - oldMean)*(value - mean);
    }

    if (count < 2) {
        return 0.0;
    }
    return Math.sqrt(stdDeviation / count);
}

@Override
public Object reset() {
}

```

```

        sum = mean = oldMean = 0.0;
        stdDeviation = 0.0;
        count = 0;
        return 0;
    }

    @Override
    public Object processAdd(Object[] data) {
        return new IllegalStateException("Stddev cannot process data array, but found
" + Arrays.deepToString(data));
    }

    @Override
    public Object processRemove(Object[] data) {
        return new IllegalStateException("Stddev cannot process data array, but found
" + Arrays.deepToString(data));
    }

    /**
     * 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() {
    }

    /**
     * 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() {
    }

    /**
     * Used to collect the serialisable 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[] {sum, mean, oldMean, stdDeviation, count};
    }

    /**
     * Used to restore serialised 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) {

```

```
sum = (Double) state[0];
mean = (Double) state[1];
oldMean = (Double) state[2];
stdDeviation = (Double) state[3];
count = (Integer) state[4];
}
```

```
}
```

Writing a Custom Stream Function Extension

To write a custom Window, create a class extending "**org.wso2.siddhi.core.query.processor.stream.window.WindowProcessor**" and create an appropriate .siddhiext extension mapping file, compile the class, and build the jar containing the .class and .siddhiext files. Add them to the Siddhi class path. In the case of running them on WSO2 DAS add the jar to <DAS_HOME>/repository/components/lib.

For example, Window extension created with namespace "custom" and function name "lastUnique" can be referred in the query as follows:

```
from StockExchangeStream[price >= 20]#window.custom:lastUnique(symbol,5)
select symbol, price
insert into StockQuote;
```

For the Window extension to be used in a Join Query the Window Extension should be findable. To make the Window findable it should implement the "**org.wso2.siddhi.core.query.processor.stream.window.FindableProcessor**" Interface.

E.g. Implementation can be found below;

```
/*
 * Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
 *
 * Licensed 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.core.query.processor.stream.window;

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.ExpressionExecutor;
import org.wso2.siddhi.core.executor.VariableExpressionExecutor;
import org.wso2.siddhi.core.query.processor.Processor;
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.expression.Expression;
```

```

import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class LastUniqueWindowProcessor extends WindowProcessor implements
FindableProcessor{
    private ConcurrentHashMap<String, StreamEvent> map = new ConcurrentHashMap<String,
StreamEvent>();
    private VariableExpressionExecutor[] variableExpressionExecutors;

    /**
     * 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) {
        variableExpressionExecutors = new
VariableExpressionExecutor[attributeExpressionExecutors.length];
        for (int i = 0; i < attributeExpressionExecutors.length; i++) {
            variableExpressionExecutors[i] =(VariableExpressionExecutor)
attributeExpressionExecutors[i];
        }
    }

    /**
     * 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) {
        ComplexEventChunk<StreamEvent> complexEventChunk = new
ComplexEventChunk<StreamEvent>();

        StreamEvent streamEvent = streamEventChunk.getFirst();
        while (streamEvent != null) {
            StreamEvent clonedEvent = streamEventCloner.copyStreamEvent(streamEvent);
            clonedEvent.setType(StreamEvent.Type.EXPIRED);

            StreamEvent oldEvent = map.put(generateKey(clonedEvent), clonedEvent);
            if (oldEvent != null) {
                complexEventChunk.add(oldEvent);
            }
            StreamEvent next = streamEvent.getNext();
            streamEvent.setNext(null);
            complexEventChunk.add(streamEvent);
            streamEvent = next;
        }
        nextProcessor.process(complexEventChunk);
    }
}

```

```

}

/**
 * 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() {
    //Do nothing
}

/**
 * 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() {
    //Do nothing
}

/**
 * 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[]{map};
}

/**
 * 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) {
    map = (ConcurrentHashMap<String, StreamEvent>) state[0];
}

/**
 * To find events from the processor event pool, that 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, map.values(), 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);
}

private String generateKey(StreamEvent event) {
    StringBuilder stringBuilder = new StringBuilder();
    for (VariableExpressionExecutor executor : variableExpressionExecutors) {
        stringBuilder.append(event.getAttribute(executor.getPosition()));
    }
    return stringBuilder.toString();
}

```

```
}
```

Writing a Custom Function

To write a custom function, create a class extending "org.wso2.siddhi.core.executor.function.FunctionExecutor", add the `SiddhiExtionsion` annotation, compile that class, and add the jar file to the class path <DAS_HOME>/repository/components/lib. Then add the **fully-qualified class name** for the implementation class in a new line, to the `siddhi.extension` file located at <DAS_HOME>/repository/conf/siddhi.

For example, if you have created the extension with namespace *custom* and function name *plus*, you can use that in the query as follows:

```
from InMediationStatsStream
select
meta_host,timestamp,resource_id,direction,fault_count,custom:plus(fault_count,count)
as totalCount
insert into OutMediationStatsStream;
```

```
import org.apache.log4j.Logger;
import org.wso2.siddhi.core.config.SiddhiContext;
import org.wso2.siddhi.core.exception.QueryCreationException;
import org.wso2.siddhi.core.executor.function.FunctionExecutor;
import org.wso2.siddhi.query.api.definition.Attribute;
import org.wso2.siddhi.query.api.extension.annotation.SiddhiExtension;

@siddhiExtension(namespace = "custom", function = "plus")
public class CustomFunctionExtension extends FunctionExecutor {
    Logger log = Logger.getLogger(CustomFunctionExtension.class);
    Attribute.Type returnType;

    /**
     * Method will be called when initialising the custom function
     *
     * @param types
     * @param siddhiContext
     */
    @Override
    public void init(Attribute.Type[] types, SiddhiContext siddhiContext) {
        for (Attribute.Type attributeType : types) {
            if (attributeType == Attribute.Type.DOUBLE) {
                returnType = attributeType;
                break;
            } else if ((attributeType == Attribute.Type.STRING) || (attributeType == Attribute.Type.BOOL)) {
                throw new QueryCreationException("Plus cannot have parameters with types String or Bool");
            } else {
                returnType = Attribute.Type.LONG;
            }
        }
    }
}

/**
```

```

 * Method called when sending events to process
 *
 * @param obj
 * @return
 */
@Override
protected Object process(Object obj) {

    if (returnType == Attribute.Type.DOUBLE) {
        double total = 0;
        if (obj instanceof Object[]) {
            for (Object aObj : (Object[]) obj) {
                total += Double.parseDouble(String.valueOf(aObj));
            }
        }
        return total;
    } else {
        long total = 0;
        if (obj instanceof Object[]) {
            for (Object aObj : (Object[]) obj) {
                total += Long.parseLong(String.valueOf(aObj));
            }
        }
        return total;
    }
}

@Override
public void destroy() {
}
/***
 * Return type of the custom function mentioned
 *
 * @return
 */
@Override
public Attribute.Type getReturnType()
    return returnType;

```

```

    }
}
```

Sample project file can be downloaded from [here](#).

Siddhi Extensions

Following are the Siddhi extensions you can use in processing events using WSO2 DAS.

- math
- str
- geo
- r
- regex
- time
- nlp
- pmml

math

Following are the functions of the Math extension.

abs

```
<double> abs(<float|double> p1)
```

- **Extension Type:** Function
- **Description:** Returns the absolute value of **p1**. This function wraps the java.lang.Math.abs() function.
- **Examples:** `abs(3)`, `abs(-3)`. Both these queries return 3, since the absolute value of both 3 and -3 is 3.

acos

```
<double> acos(<float|double> p1)
```

- **Extension Type:** Function
- **Description:** Returns the arc-cosine (inverse cosine) of **p1** if $-1 \leq p1 \leq 1$, or returns NULL otherwise. The return value is in radian scale. This function wraps the java.lang.Math.acos() function.
- **Example:** `acos(0.5)` returns 1.0471975511965979.

asin

```
<double> asin (<float|double> p1)
```

- **Extension Type:** Function
- **Description:** Returns the arc-sin (inverse sine) of **p1** if $-1 \leq p1 \leq 1$, or returns NULL otherwise. The return value is in radian scale. This function wraps the java.lang.Math.asin() function.
- **Example:** `asin(0.5)` returns 0.5235987755982989.

atan

```
<double> atan(<int|long|float|double> p1)
```

- **Extension Type:** Function
- **Description:** Returns the arc-tangent (inverse tangent) of **p1**. The return value is in radian scale. This function wraps the java.lang.Math.atan() function.
- **Examples:** `atan(6d)` returns 1.4056476493802699.

```
<double> atan (<int|long|float|double> p1, <int|long|float|double> p2)
```

- **Extension Type:** Function
- **Description:** Returns the arc-tangent (inverse tangent) of coordinates **p1** and **p2**. The return value is in radian scale. This function wraps the java.lang.Math.atan2() function.
- **Parameter:** **p1**: Ordinate coordinate.
- **Parameter:** **p2**: Abscissa coordinate.
- **Examples:** `atan(12d, 5d)` returns 1.1760052070951352.

bin

<string> bin(<int|long> p1)

- **Extension Type:** Function
- **Description:** Returns a string representation of the integer/long argument **p1**, as an unsigned integer in base 2. This function wraps the java.lang.Integer.toBinaryString and java.lang.Long.toBinaryString methods.
- **Example:** bin(9) returns "1001".

ceil

<double> ceil(<float|double> p1)

- **Extension Type:** Function
- **Description:** Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument **p1**, and is equal to a mathematical integer. Wraps java.lang.Math.ceil() method.
- **Example:** ceil(423.187d) returns 424.0.

conv

<string> conv(<string> a, <int> fromBase, <int> toBase)

- **Extension Type:** Function
- **Description:** Converts **a** from base **fromBase** to base **toBase**.
- **Example:** conv("7f", 16, 10) returns "127".

copySign

<double> copySign(<int|long|float|double> magnitude, <int|long|float|double> sign)

- **Extension Type:** Function
- **Description:** Returns the magnitude of **magnitude** with the sign of **sign**. This function wraps the java.lang.Math.copySign() function.
- **Example:** copySign(5.6d, -3.0d) returns -5.6.

cos

<double> cos(<int|long|float|double> p1)

- **Extension Type:** Function
- **Description:** Returns the cosine of **p1** (**p1** is in radians). This function wraps the java.lang.Math.cos() function.
- **Example:** cos(6d) returns 0.9601702866503661.

cosh

<double> cosh(<int|long|float|double> p1)

- **Extension Type:** Function
- **Description:** Returns the hyperbolic cosine of **p1** (**p1** is in radians). This function wraps the java.lang.Math.cosh() function.
- **Example:** cosh(6d) returns 201.7156361224559.

cbrt

<double> cbrt(<int|long|float|double> p1)

- **Extension Type:** Function
- **Description:** Returns the cube-root of **p1** (**p1** is in radians). This function wraps the java.lang.Math.cbrt() function.
- **Example:** cbrt(17d) returns 2.5712815906582356.

e

<double> e()

- **Extension Type:** Function
- **Description:** Returns the constant java.lang.Math.E, which is the double value that is closer than any other to e, which is the base of the natural logarithms.
- **Example:** e() always return 2.7182818284590452354.

exp

<double> **exp**(<int|long|float|double> **p1**)

- **Extension Type:** Function
- **Description:** Returns Euler's number e raised to the power of **p1**. This function wraps the `java.lang.Math.exp()` function.
- **Example:** `exp(10.23)` returns 27722.51006805505.

`floor`

<double> **floor**(<int|long|float|double> **p1**)

- **Extension Type:** Function
- **Description:** This function wraps the `java.lang.Math.floor()` function which returns the largest (closest to positive infinity) value that is less than or equal to **p1** and is equal to a mathematical integer.
- **Example:** `floor(10.23)` returns 10.0.

`getExponent`

<double> **getExponent**(<int|long|float|double> **p1**)

- **Extension Type:** Function
- **Description:** Returns the unbiased exponent used in the representation of **p1**. This function wraps the `java.lang.Math.getExponent()` function.
- **Example:** `getExponent(60984.1)` returns 15.

`hex`

<string> **hex**(<int|long|float|double> **p1**)

- **Extension Type:** Function
- **Description:** This function wraps the `java.lang.Double.toHexString()` function which returns a hexadecimal string representation of **p1**.
- **Example:** `hex(200)` returns "c8".

`isInfinite`

<boolean> **isInfinite**(<float|double> **p1**)

- **Extension Type:** Function
- **Description:** This function wraps the `java.lang.Float.isInfinite()` and `java.lang.Double.isInfinite()` functions which returns true, if **p1** is infinitely large in magnitude, or returns false otherwise.
- **Example:** `isInfinite(java.lang.Double.POSITIVE_INFINITY)` returns true.

`isNaN`

<boolean> **isNaN**(<float|double> **p1**)

- **Extension Type:** Function
- **Description:** This function wraps the `java.lang.Float isNaN()` and `java.lang.Double isNaN()` functions which returns true if **p1** is a Not-a-Number (NaN) value, or returns false otherwise.
- **Example:** `isNaN(java.lang.Math.log(-12d))` returns true.

`In`

<double> **ln** (< int|long|float|double > **p1**)

- **Extension Type:** Function
- **Description:** Returns the natural logarithm (base e) of **p1**.
- **Example:** `ln(11.453)` returns 2.438251704415579.

`log2`

<double> **log2** (< int|long|float|double > **p1**)

- **Extension Type:** Function
- **Description:** Returns the base 2 logarithm of **p1**.
- **Example:** `log2(91d)` returns 6.507794640198696.

`log10`

<double> **log10** (< int|long|float|double > **p1**)

- **Extension Type:** Function
- **Description:** Returns the base 10 logarithm of **p1**.
- **Example:** `log10(19.234)` returns 1.2840696117100832

`log``<double> log (< int|long|float|double > number, < int|long|float|double > base)`

- **Extension Type:** Function
- **Description:** Returns the logarithm (base=**base**) of **number**.
- **Example:** `log(34, 2f)` returns 5.08746284125034.

`max``<double> max (< int|long|float|double > p1, < int|long|float|double > p2)`

- **Extension Type:** Function
- **Description:** Returns the greater of **p1** and **p2**.
- **Example:** `max(123.67d, 91)` returns 123.67.

`min``<double> min (< int|long|float|double > p1, < int|long|float|double > p2)`

- **Extension Type:** Function
- **Description:** Returns the smaller of **p1** and **p2**.
- **Example:** `min(123.67d, 91)` returns 91.

`oct``<string> oct (<int|long> p1)`

- **Extension Type:** Function
- **Description:** Converts **p1** to octal.
- **Example:** `oct(991)` returns "143".

`parseDouble``<double> parseDouble (<string> str)`

- **Extension Type:** Function
- **Description:** Returns **str** as a double.
- **Example:** `parseDouble("123")` returns 123.0.

`parseFloat``<float> parseFloat (<string> str)`

- **Extension Type:** Function
- **Description:** Returns **str** as a float.
- **Example:** `parseFloat("123")` returns 123.0.

`parseInt``<int> parseInt (<string> str)`

- **Extension Type:** Function
- **Description:** Returns **str** as an int.
- **Example:** `parseInt("123")` returns 123.

`parseLong``<long> parseLong (<string> str)`

- **Extension Type:** Function
- **Description:** Returns **str** as a long.
- **Example:** `parseLong("123")` returns 123.

`pi``<double> pi ()`

- **Extension Type:** Function
- **Description:** Returns the constant `java.lang.Math.PI`, which is the value that is closer than any other to pi, the

ratio of the circumference of a circle to its diameter.

- **Example:** `pi()` always return 3.141592653589793.

`power`

```
<double> power ( < int|long|float|double> value, <int|long|float|double> toPower )
```

- **Extension Type:** Function

- **Description:** Returns `value` raised to the power of `toPower`.

- **Example:** `power(5.6d, 3.0d)` returns 175.61599999999996.

`rand`

```
<double> rand ( )
```

- **Extension Type:** Function

- **Description:** A sequence of calls to `rand()` generates a stream of pseudo-random numbers. This function internally uses `java.util.Random` class.

- **Example:** Two sequential calls to `rand()` may return 0.8263929447650588 and 0.24425883860361197, respectively.

```
<double> rand (< int|long > seed)
```

- **Extension Type:** Function

- **Description:** A sequence of calls to `rand(seed)` generates a stream of pseudo-random numbers. This function internally uses `java.util.Random` class.

- **Parameter:** `seed`: This is the initial seed given when the random number generation starts.

- **Example:** Two sequential calls to `rand(12)` may return 0.7298928061101974 and 0.2750691655200749, respectively.

`round`

```
<int> round (<float> value )
```

- **Extension Type:** Function

- **Description:** Returns the closest integer value to the argument

- **Example:** `round(3.35)` will return 3

```
<long> round (<double> value )
```

- **Extension Type:** Function

- **Description:** Returns the closest long value to the argument

- **Example:** `round(3252.353)` will return 3252

`signum`

```
<int> signum (< int|long|float|double > p1)
```

- **Extension Type:** Function

- **Description:** Returns the sign of `p1` as '1.0' (if `a` is positive) or '-1.0' (if `a` is negative), '0.0' otherwise. This function wraps `java.lang.Math.signum()` function.

- **Example:** `signum(-6.32d)` returns -1.

`sin`

```
<double> sin (< int|long|float|double > p1)
```

- **Extension Type:** Function

- **Description:** Returns the sine of `p1` (`p1` is in radians). This function wraps `java.lang.Math.sin()` function.

- **Example:** `sin(6d)` returns -0.27941549819892586.

`sinh`

```
<double> sinh (< int|long|float|double > p1)
```

- **Extension Type:** Function

- **Description:** Returns the hyperbolic sine of `p1` (`p1` is in radians). This function wraps `java.lang.Math.sinh()` function.

- **Example:** `sinh(6d)` returns 201.71315737027922.

sqrt

```
<double> sqrt (< int|long|float|double > p1)
```

- **Extension Type:** Function

- **Description:** Returns the square-root of **p1**. This function wraps `java.lang.Math.sqrt()` function.

- **Example:** `sqrt(4d)` returns 2.

tan

```
<double> tan (< int|long|float|double > p1)
```

- **Extension Type:** Function

- **Description:** Returns the tan of **p1** (**p1** is in radians). This function wraps `java.lang.Math.tan()` function.

- **Example:** `tan(6d)` returns -0.29100619138474915.

tanh

```
<double> tanh (<int|long|float|double> p1)
```

- **Extension Type:** Function

- **Description:** Returns the hyperbolic tangent of **p1** (**p1** is in radians). This function wraps `java.lang.Math.tanh()` function.

- **Example:** `tanh(6d)` returns 0.9999877116507956.

toDegrees

```
<double> toDegrees (< int|long|float|double > p1)
```

- **Extension Type:** Function

- **Description:** Converts **p1** from radians to degrees. This function wraps `java.lang.Math.toDegrees()` function.

- **Example:** `toDegrees(6d)` returns 343.77467707849394.

toRadians

```
<double> toRadians (< int|long|float|double > p1)
```

- **Extension Type:** Function

- **Description:** Converts **p1** from degrees to radians. This function wraps `java.lang.Math.toRadians()` function.

- **Example:** `toRadians(6d)` returns 0.10471975511965977.

str

Following are the functions of the String extension.

charAt

```
<string> charAt(<string> str, <int> index)
```

- **Extension Type:** Function

- **Description:** Returns the char value in **str** at the specified **index**.

- **Examples:** `charAt("WSO2", 1)` returns 'S'

coalesce

```
< 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 function accepts any number of parameters and they can be of different types too.

- **Return Type :** Return type will be the type of the first of its input parameters that is not null.

- **Examples:** `coalesce("123", null, "789")` returns "123". `coalesce(null, "BBB", "CCC")` returns "BBB". `coalesce(null, null, null)` returns null.

concat

```
<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:** concat("D533", "8JU^", "XYZ") returns "D5338JU^XYZ". concat("AAA", null, "CCC") returns "AAACCC".

hex

<string> **hex** (< string> **str**)

- **Extension Type:** Function
- **Description:** Returns a hexadecimal string representation of **str**
- **Example:** hex("MySQL") returns "4d7953514c".

length

<int> **length** (< string> **str**)

- **Extension Type:** Function
- **Description:** Returns the length of the string: **str**.
- **Example:** length("Hello World") returns 11.

lower

<string> **lower** (< string> **str**)

- **Extension Type:** Function
- **Description:** Converts the capital letters in the input string: **str**, to the equivalent simple letters.
- **Example:** lower("WSO2 DAS") returns "wso2 DAS".

regexp

<boolean> **regexp** (< string> **str**, <string> **regex**)

- **Extension Type:** Function
- **Description:** Tells whether or not the string: **str**, matches the given regular expression: **regex**.
- **Example:** regexp("WSO2 abcdh" , "WSO(.*)h") returns true.

repeat

<string> **repeat** (< string> **str**, <int> **times**)

- **Extension Type:** Function
- **Description:** Repeats the string: **str**, for a specified number of times: **times**.
- **Example:** repeat("StRing 1" , 3) returns "StRing 1StRing 1StRing 1".

replaceAll

<string> **replaceAll** (< string> **str**, <string> **regex** , <string> **replacement**)

- **Extension Type:** Function
- **Description:** Replaces each substring of **str** that matches the given **regex** with the given **replacement**.
- **Example:** replaceAll("hello hi hello" , 'hello' , 'test') returns "test hi test".

replaceFirst

<string> **replaceFirst** (< string> **str** , <string> **regex** , <string> **replacement**)

- **Extension Type:** Function
- **Description:** Replaces the first substring of **str** that matches the given **regex** with the given **replacement**.
- **Example:** replaceFirst("hello WSO2 A hello" , 'WSO2(.*)A' , 'XXXX') returns "hello XXXX hello"

reverse

<string> **reverse** (< string> **str**)

- **Extension Type:** Function
- **Description:** Returns the reverse ordered string of **str**.
- **Example:** reverse("Hello World") returns "dlroW olleH".

strcmp

<int> **strcmp** (< string> **str**, <string> **compareTo**)

- **Extension Type:** Function
- **Description:** Compares **str** with **compareTo** strings lexicographically.
- **Examples:** `strcmp("Hello", 'Hello')` returns 0. `strcmp("AbCDefghiJ KLMN", 'Hello')` returns -7.

substr

```
<string> substr ( <string> sourceText, <int> beginIndex )
```

- **Extension Type:** Function
- **Description:** Returns a new string that is a substring of **sourceText**.
- **Example:** `substr("AbCDefghiJ KLMN", 4)` returns "efghiJ KLMN".

```
<string> substr ( <string> sourceText, <int> beginIndex, <int> length )
```

- **Extension Type:** Function
- **Description:** Returns a new string that is a substring of **sourceText**.
- **Example:** `substr("AbCDefghiJ KLMN", 2, 4)` returns "CDef".

```
<string> substr ( <string> sourceText, <string> regex)
```

- **Extension Type:** Function
- **Description:** Returns a new string that is a substring of **sourceText**.
- **Examples:** `substr("WSO2D efghiJ KLMN", '^WSO2(.*)')` returns "WSO2D efghiJ KLMN".

```
<string> substr ( <string> sourceText, <string> regex, <int> groupNumber )
```

- **Extension Type:** Function
- **Description:** Returns a new string that is a substring of **sourceText**.
- **Example:** `substr("WSO2 DAS WSO2 XX E hi hA WSO2 heAllo", 'WSO2(.*)A(.*)', 2)` return s "ello".

trim

```
<string> trim ( <string> str)
```

- **Extension Type:** Function
- **Description:** Returns a copy of **str**, with leading and trailing white-spaces omitted.
- **Example:** `trim(" AbCDefghiJ KLMN ")` returns "AbCDefghiJ KLMN".

unhex

```
<string> unhex ( <string> str)
```

- **Extension Type:** Function
- **Description:** This is the equivalent of 'unhex' function in mysql 5.0. `unhex(str)` interprets each pair of characters in **str** as a hexadecimal number. Also see `hex()` string-extension in Siddhi.
- **Example:** `unhex("4d7953514c")` returns "MySQL".

upper

```
<string> upper ( <string> str)
```

- **Extension Type:** Function
- **Description:** Converts the simple letters in the input string: **str** to the equivalent capital letters.
- **Example:** `upper("Hello World")` returns "HELLO WORLD"

contains

```
<bool> contains ( <string> inputSequence, <string> searchingSequence )
```

- **Extension Type:** Function
- **Description:** method returns true if and only if the **inputSequence** contains the specified sequence of char values in the **searchingSequence**.
- **Example:** `contains("21 products are produced by WSO2 currently", "WSO2")` returns true

geo

Following are the functions of the Geo extension.

intersects

```
<bool> intersects (<string> geoJSONGeometry , <string> geoJSONGeometryFence )
```

- **Extension Type:** Function
- **Description:** Returns true if the incoming event geoJSONGeometry intersects the given geoJSONGeometryFence else false.
- **Example:** intersects({'type':'Polygon','coordinates':[[[0.5, 0.5],[0.5, 1.5],[1.5, 1.5],[1.5, 0.5],[0.5, 0.5]]]} , {'type':'Polygon','coordinates':[[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]]}), returns true because geoJSONGeometry intersects geoJSONGeometryFence.

```
<bool> intersects (<double> longitude , <double> latitude , <string> geoJSONGeometryFence )
```

- **Extension Type:** Function
- **Description:** Returns true if the location pointed by longitude and latitude intersects the given geoJSONGeometryFence else false.
- **Example:** intersects(0.5, 0.5 , {'type':'Polygon','coordinates':[[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]]}), returns true because location pointed by longitude and latitude intersects geoJSONGeometryFence .

within

```
<bool> within (<double> longitude , <double> latitude , <string> geoJSONGeometryFence )
```

- **Extension Type:** Function
- **Description:** Returns true if the location pointed by longitude and latitude is within the geoJSONGeometryFence else false.
- **Examples:** within(0.5, 0.5, {'type':'Polygon','coordinates':[[[0,0],[0,2],[1,2],[1,0],[0,0]]]}) returns true while within(2, 2, {'type':'Polygon','coordinates':[[[0,0],[0,2],[1,2],[1,0],[0,0]]]}) returns false.

```
<bool> within (<string> geoJSONGeometry , <string> geoJSONGeometryFence )
```

- **Extension Type:** Function
- **Description:** Returns true if geoJSONGeometry is within the geoJSONGeometryFence else false.
- **Examples:** 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 while, 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

```
<bool> withindistance (<double> longitude , <double> latitude , <string> geoJSONGeometryFence )
```

- **Extension Type:** Function
- **Description:** Returns true if the location given by longitude and latitude is within distance of the geoJSONGeometryFence else false.
- **Example:** withindistance(0.5 , 0.5, {'type':'Polygon','coordinates':[[[0, 0],[0, 1],[1, 1],[1, 0],[0, 0]]]}, 110574.61087757687) returns true because location given by longitude and latitude is within distance of geoJSONGeometryFence.

```
<bool> withindistance (<string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> distance )
```

- **Extension Type:** Function
- **Description:** Returns true if the area given by geoJSONGeometry is within distance of the geoJSONGeometryFence else false.
- **Example:** 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) returns true because geoJSONGeometry is within distance of geoJSONGeometryFence.

crosses

```
<bool> crosses (<string> id , <double> longitude, <double> latitude , <string> geoJSONGeometryFence )
```

- **Extension Type:** StreamProcessor
- **Description:** An event with 'crosses' additional attribute set to true when the object ((longitude, latitude)) crosses into geoJSONGeometryFence and an event with `crosses` additional attribute set to false when the object crosses out of the geoJSONGeometryFence.
- **Example:** crosses(km-4354, -0.5, 0.5, {`type`:'Polygon','coordinates':[[[0, 0],[2, 0],[2, 1],[0, 1],[0, 0]]]}) and crosses(km-4354, 1.5, 0.5, {`type`:'Polygon','coordinates':[[[0, 0],[2, 0],[2, 1],[0, 1],[0, 0]]]}) return true because these geo locations crosses each other.

```
<bool> crosses (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence )
```

- **Extension Type:** StreamProcessor
- **Description:** An event with 'crosses' additional attribute set to true when the object (geoJSONGeometry) crosses into geoJSONGeometryFence and an event with `crosses` additional attribute set to false when the object crosses out of the geoJSONGeometryFence.
- **Example:**

stationary

```
<bool> stationary (<string> id , <double> longitude, <double> latitude , <string> geoJSONGeometryFence , <double> radius )
```

- **Extension Type:** StreamProcessor
- **Description:** When the object (longitude, latitude) starts being stationary within the radius an event with 'stationary' additional attribute set to true. When the object starts to move out of the radius an event with 'stationary' additional attribute set to false.
- **Example:** stationary(km-4354,0,0, 110574.61087757687) and stationary(km-4354,1,1, 110574.61087757687) and stationary(km-4354,1,1.5, 110574.61087757687) return true.

```
<bool> stationary (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> radius )
```

- **Extension Type:** StreamProcessor
- **Description:** When the object (geoJSONGeometry) starts being stationary within the radius an event with 'stationary' additional attribute set to true. When the object starts to move out of the radius an event with 'stationary' additional attribute set to false.
- **Example:**

proximity - validate this

```
<bool> proximity (<string> id , <double> longitude, <double> latitude , <string> geoJSONGeometryFence , <double> radius )
```

- **Extension Type:** StreamProcessor
- **Description:** When two objects (longitude, latitude) starts being in close proximity within the radius an event with 'inCloseProximity' additional attribute set to true. When the object starts to move out of the radius an event with 'inCloseProximity' additional attribute set to false. On each event, additional attributes 'proximityWith' gives the id of the object that this object is in close proximity and 'proximityId' is an id unique to the pair of objects.
- **Example:** proximity(1, 0, 0, 110574.61087757687) and proximity(2, 1, 1, 110574.61087757687) and proximity(3, 2, 2, 110574.61087757687) and proximity(1, 1.5, 1.5, 110574.61087757687) returns true with ID 3.

```
<bool> proximity (<string> id , <string> geoJSONGeometry , <string> geoJSONGeometryFence , <double> radius )
```

- **Extension Type:** StreamProcessor
- **Description:** When two objects (geoJSONGeometry) starts being in close proximity within the radius an event with 'inCloseProximity' additional attribute set to true. When the object starts to move out of the radius an event with 'inCloseProximity' additional attribute set to false. On each event, additional attributes

'proximityWith' gives the id of the object that this object is in close proximity and 'proximityId' is an id unique to the pair of objects.

- **Example:**

geocode

```
<double, double, string> geocode (<string> location )
```

- **Extension Type:** StreamProcessor
- **Description:** transforms a location into its geo-coordinates (longitude and latitude) and formatted address
- **Example:** geocode(duplication rd) returns the following data 6.8995244d, 79.8556202d, "R A De Mel Mawatha, Colombo, Sri Lanka" with adhering latitude, longitude, formattedAddress attribute names respectively.

r

Following are the functions of the R extension.

eval

```
[<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 , ... )
```

- **Extension Type:** Stream Processor
- **Description:** This will run the R script for each event and produce aggregated outputs based on the provided input variable parameters and expected output attributes.
- **Parameter:** **script**: R script as a string that uses the input variable parameters and produce the expected output attributes.
- **Parameter:** **outputAttributes**: All output attributes separated by comma as string here each attribute is denoted as <name><space><type>. e.g., 'output1 string, output2 long'
- **Parameter:** **input1, input2, ...**: Input parameters have to be variable attributes of the input stream, function does not accept any constant values as input parameters.
- **Return Parameter:** **output1, output2, ...** : Output parameters will be generated according to the provided **outputAttributes**.
- **Examples** : eval('totalItems <- sum(items); totalTemp <- sum(temp);', 'totalItems int, totalTemp double', items, temp) where items being int and temp being double will return [totalItems, totalTemp] where totalTemp will be int and totalTemp will be double.

evalSource

```
[<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 will run the R script loaded from a file for each event and produce aggregated outputs based on the provided input variable parameters and expected output attributes.
- **Parameter:** **filePath**: The file path of R script where this script that uses the input variable parameters and produce the expected output attributes.
- **Parameter:** **outputAttributes**: All output attributes separated by comma as string here each attribute is denoted as <name><space><type>. e.g., 'output1 string, output2 long'
- **Parameter:** **input1, input2, ...**: Input parameters have to be variable attributes of the input stream, function does not accept any constant values as input parameters.
- **Return Parameter:** **output1, output2, ...** : Output parameters will be generated according to the provided **outputAttributes**.
- **Examples:** eval('/home/user/test/script1.R', 'totalItems int, totalTemp double', items, temp) where items being int and temp being double will return [totalItems, totalTemp] where totalTemp will be int and totalTemp will be double.

regex

Following are the functions of the RegEx extension.

find

```
<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 and returns true if the sub sequence exists else false
- **Examples:** `find("\d\d(.*)WSO2", "21 products are produced by WSO2 currently")` returns true while `find("\d\d(.*)WSO2", "21 products are produced currently")` returns false.

```
<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 a given index in the 'inputSequence' and returns true if the sub sequence exists else false
- **Examples:** `find("\d\d(.*)WSO2", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 30)` returns true while `find("\d\d(.*)WSO2", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 35)` returns false.

group

```
<string> group (<string> regex , <string> inputSequence , <int> groupId )
```

- **Extension Type:** Function
- **Description:** This method returns the input sub-sequence captured by the given group during the previous match operation else returns null
- **Examples:** `group("(\\d\\d)(.*)(WSO2.*)", "21 products are produced within 10 years by WSO2 currently by WSO2 employees", 3)` returns "WSO2 employees".

lookingAt

```
<string> lookingAt (<string> regex , <string> inputSequence )
```

- **Extension Type:** Function
- **Description:** This method attempts to match the 'inputSequence', starting at the beginning, against the 'regex' pattern.
- **Examples:** `lookingAt("\d\\d(.*)WSO2", "21 products are produced by WSO2 currently in Sri Lanka")` returns true while `lookingAt("WSO2(.*)middleware(.*)", "sample test string and WSO2 is situated in trace and its a middleware company")` returns false.

matches

```
<string> matches (<string> regex , <string> inputSequence )
```

- **Extension Type:** Function
- **Description:** This method attempts to match the entire 'inputSequence' against the 'regex' pattern.
- **Examples:** `matches("WSO2(.*)middleware(.*)", "WSO2 is situated in trace and its a middleware company")` returns true while `matches("WSO2(.*)middleware", "WSO2 is situated in trace and its a middleware company")` returns false.

time

Following are the functions of the time extension.

currentDate

```
<string> currentDate ( )
```

- **Extension Type:** Function
- **Description:** This method returns current system date in yyyy-MM-dd format.
- **Examples:** `currentTime()` returns 2015-08-20.

currentTime

```
<string> currentTime ( )
```

- **Extension Type:** Function
- **Description:** This method returns current system time in HH:mm:ss format.
- **Examples:** `currentTime()` returns 13:15:10.

currentTimestamp

```
<string> currentTimestamp ( )
```

- **Extension Type:** Function
- **Description:** This method returns current system timestamp in yyyy-MM-dd HH:mm:ss format.
- **Examples:** currentTime() returns 2015-08-20 13:15:10.

dateAdd

- dateValue - value of date. eg: "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- expr - In which amount, selected date format part should be incremented. eg: 2 ,5 ,10 etc
- unit - Which part of the date format you want to manipulate. eg: "MINUTE" , "HOUR" , "MONTH" , "YEAR" , "QUARTER" , * "WEEK" , "DAY" , "SECOND"
- dateFormat - Date format of the provided date value. eg: yyyy-MM-dd HH:mm:ss.SSS
- timestampInMilliseconds - date value in milliseconds.(from the epoch) eg: 1415712224000L

```
<string> dateAdd (<string> dateValue , <long> expr, <string> unit, <string> dateFormat )
```

- **Extension Type:** Function
- **Description:** this method returns added specified time interval to a date.
- **Examples:** dateAdd("2014-11-11 13:23:44", 2, 'year','yyyy-MM-dd HH:mm:ss') will return "2016-11-11 13:23:44"

```
<string> dateAdd (<string> dateValue , < long > expr, <string> unit )
```

- **Extension Type:** Function
- **Description:** this method returns added specified time interval to a date.
- **Examples:** dateAdd("2014-11-11 13:23:44", 2, 'year') will return "2016-11-11 13:23:44"

```
<string> dateAdd (<long> timestampInMilliseconds, < long > expr, <string> unit )
```

- **Extension Type:** Function
- **Description:** this method returns added specified time interval to a timestamp in milliseconds.
- **Examples:** dateAdd(1415692424000L, 2, 'year') will return "2016-11-11 13:23:44"

dateSub

- dateValue - value of date. eg: "2014-11-11 13:23:44.657", "2014-11-11", "13:23:44.657"
- unit - Which part of the date format you want to manipulate. eg: "MINUTE" , "HOUR" , "MONTH" , "YEAR" , "QUARTER" , * "WEEK" , "DAY" , "SECOND"
- expr - In which amount, selected date format part should be decremented. eg: 2 ,5 ,10 etc
- dateFormat - Date format of the provided date value. eg: yyyy-MM-dd HH:mm:ss.SSS
- timestampInMilliseconds - date value in milliseconds.(from the epoch) eg: 1415712224000L

```
<string> dateSub (<string> dateValue , <long> expr, <string> unit, <string> dateFormat )
```

- **Extension Type:** Function
- **Description:** this method returns added specified time interval to a date.
- **Examples:** dateSub("2014-11-11 13:23:44", 2, 'year','yyyy-MM-dd HH:mm:ss') will return "2012-11-11 13:23:44"

```
<string> dateSub (<string> dateValue , < long > expr, <string> unit )
```

- **Extension Type:** Function
- **Description:** this method returns added specified time interval to a date.
- **Examples:** dateSub("2014-11-11 13:23:44", 2, 'year') will return "2012-11-11 13:23:44"

```
<string> dateSub (<long> timestampInMilliseconds, < long > expr, <string> unit )
```

- **Extension Type:** Function
- **Description:** this method returns added specified time interval to a timestamp in milliseconds.
- **Examples:** dateSub(1415692424000L, 2, 'year') will return 1352620424000

dateDiff

- dateValue1 - value of date. eg: "2014-11-11 13:23:44.657", "2014-11-11" , "13:23:44.657"
- dateValue2 - value of date. eg: "2014-11-11 13:23:44.657", "2014-11-11" , "13:23:44.657"
- dateFormat1 - Date format of the provided dateValue1. eg: yyyy-MM-dd HH:mm:ss.SSS
- dateFormat2 - Date format of the provided dateValue2. eg: yyyy-MM-dd HH:mm:ss.SSS
- timestampInMilliseconds1 - date value in milliseconds.(from the epoch) eg: 1415712224000L
- timestampInMilliseconds2 - date value in milliseconds.(from the epoch) eg: 1423456224000L

```
<int> dateDiff (<string> dateValue1 , < string > dateValue2 , <string> dateFormat1  
, <string> dateFormat2 )
```

- **Extension Type:** Function
- **Description:** Returns time(days) between two dates.
- **Examples:** 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') will return 2

```
<int> dateDiff (<string> dateValue1 , < string > dateValue2 )
```

- **Extension Type:** Function
- **Description:** Returns time(days) between two dates.
- **Examples:** dateDiff('2014-11-11 13:23:44.000', '2014-11-9 13:23:44.000') will return 2

```
<int> dateDiff (<string> timestampInMilliseconds1 , < string > timestampInMilliseconds2 )
```

- **Extension Type:** Function
- **Description:** Returns time(days) between two dates.
- **Examples:** dateDiff(1415692424000, 1415519624000) will return 2

dateFormat

- dateValue - value of date. eg: "2014-11-11 13:23:44.657", "2014-11-11" , "13:23:44.657"
- dateTargetFormat - Date format which need to be converted to. eg: yyyy/MM/dd HH:mm:ss
- dateSourceFormat - Date format of the provided date value. eg: yyyy-MM-dd HH:mm:ss.SSS
- timestampInMilliseconds - date value in milliseconds.(from the epoch) eg: 1415712224000L
- dateTargetFormat - Date format which need to be converted to. eg: yyyy/MM/dd HH:mm:ss

```
<string> dateFormat(<string> dateValue,<string> dateTargetFormat,<string> dateSourceFormat)
```

- **Extension Type:** Function
- **Description:** Returns a formatted date string
- **Examples:** dateFormat('2014-11-11 13:23:55', 'ss', 'yyyy-MM-dd HH:mm:ss') will return 55

```
<string> dateFormat(<string> dateValue,<string> dateTargetFormat)
```

- **Extension Type:** Function
- **Description:** Returns a formatted date string
- **Examples:** dateFormat('2014-11-11 13:23:55.657', 'ss') will return 55

```
<string> dateFormat (<long> timestampInMilliseconds ,<string> dateTargetFormat)
```

- **Extension Type:** Function
- **Description:** Returns a formatted date string
- **Examples:** dateFormat(1415692424000, 'ss') will return -19756000

extract

- dateValue - value of date. eg: "2014-11-11 13:23:44.657", "2014-11-11" , "13:23:44.657"
- unit - Which part of the date format you want to manipulate. eg: "MINUTE" , "HOUR" , "MONTH" , "YEAR" , "QUARTER" , * "WEEK" , "DAY" , "SECOND"
- dateFormat - Date format of the provided date value. eg: yyyy-MM-dd HH:mm:ss.SSS
- timestampInMilliseconds - date value in milliseconds.(from the epoch) eg: 1415712224000L

```
<string> extract (<string> unit ,<string> dateValue, <string> dateFormat)
```

- **Extension Type:** Function

- **Description:** This method returns the time component of the dateValue specified in unit parameter
- **Examples:** extact('year', '2014-3-11 02:23:44', 'yyyy-MM-dd hh:mm:ss') will return 2014

<string> **extract** (<string> **unit** ,<string> **dateValue**)

- **Extension Type:** Function
- **Description:** This method returns the time component of the dateValue specified in unit parameter
- **Examples:** extact('year', '2014-3-11 02:23:44.234') will return 2014

<string> **extract** (<long> **timestampInMilliseconds** ,<string> **unit**)

- **Extension Type:** Function
- **Description:** This method returns the time component of the dateValue specified in unit parameter
- **Examples:** extact(1394484824000, 'year') will return 2014

date

<string> **date** (<string> **dateValue** ,<string> **dateFormat**)

- **Extension Type:** Function
- **Description:** This method returns the date component of the dateValue
- **Examples:** extact('2014-11-11 13:23:44', 'yyyy-MM-dd HH:mm:ss') will return 2014-11-11

timestampInMilliseconds

<long> **timestampInMilliseconds** ()

- **Extension Type:** Function
- **Description:** This method returns the current timestamp in milliseconds
- **Examples:** timestampInMilliseconds() will return 1440160328693

<long> **timestampInMilliseconds** (<string> **dateValue**)

- **Extension Type:** Function
- **Description:** This method returns the timestamp of the value specified in dateValue parameter. The value should be in 'yyyy-MM-dd HH:mm:ss.SSS' format
- **Examples:** timestampInMilliseconds('2007-11-30 10:30:19.000') will return 1196398819000

<long> **timestampInMilliseconds** (<string> **dateValue**, <string> **dateFormat**)

- **Extension Type:** Function
- **Description:** This method returns the timestamp of the value specified in dateValue parameter. The date format can be given in the dateFormat parameter
- **Examples:** timestampInMilliseconds('2007-11-30 10:30:19', 'yyyy-MM-dd HH:mm:ss') will return 1196398819000

utcTimestamp

<string> **utcTimestamp**()

- **Extension Type:** Function
- **Description:** Returns System time in yyyy-MM-dd HH:mm:ss format.
- **Examples:** utcTimestamp() will return 2015-08-21 12:16:13

nlp

findNameEntityType

<string> **findNameEntityType**(<string> **entityType**, <bool> **groupSuccessiveMatch**, <string> **string-variable**)

- **Extension Type:** Function
- **Description:**
 - The findNameEntityType function takes in
 - entityType: a user given string constant as entity Type - PERSON, LOCATION, ORGANIZATION, MONEY, PERCENT, DATE or TIME
 - groupSuccessiveMatch : user given boolean constant in order to group successive matches of the given entity type and a text stream.

- streamAttribute: a string or the stream attribute which the text stream resides
- It returns the entities in the text. If we give group successive matches as true the result will aggregate successive words of the same entity type.
- **Examples:** `findNameEntityType("PERSON",true,text)`, if text attribute contains "Bill Gates donates £31million to fight Ebola" result will be "Bill Gates". If groupSuccessiveMatch is "false" two events will be generated as "Bill" and "Gates".

`findNameEntityTypeViaDictionary`

```
<string> findNameEntityTypeViaDictionary(<string> entityType, <bool> dictionaryFilePath, <string> string-variable )
```

- **Extension Type:** Function

- **Description:**

- The `findNameEntityType` function takes in
 - entityType: a user given string constant as entity Type - PERSON, LOCATION, ORGANIZATION, MONEY, PERCENT, DATE or TIME
 - dictionaryFilePath : path to the dictionary which expected entities for the entity types and the dictionary should be in the following form

? Unknown Attachment

- string-variable: a string or the stream attribute which the text stream resides
- It returns the entities in the text. If we give group successive matches as true the result will aggregate successive words of the same entity type.
- **Examples:** `findNameEntityType("PERSON",true,text)`, if text attribute contains "Bill Gates donates £31million to fight Ebola" result will be "Bill Gates". If groupSuccessiveMatch is "false" two events will be generated as "Bill" and "Gates".

`findRelationshipByVerb`

```
<string> text, <string> subject, <string> object <string> verb findRelationshipByVerb (<string> verb, <string> string-variable )
```

- **Extension Type:** Function

- **Description:** takes in a user given string constant as a verb, and a text stream. Then it returns whole text, subject, object, verb relationship from the text stream that can be extracted for any form of that verb

- The `findRelationshipByVerb` function takes in
 - verb: user given string constant
 - string-variable : a string or the stream attribute which the text stream resides
- It returns the complete string, subject object and the verb if the entered verb is resides in the input text.

- **Examples:** `findRelationshipByVerb("say", "Information just reaching us says another Liberian With Ebola Arrested At Lagos Airport")`, returns 4 parameters. the whole text, subject as *Information*, object as *Liberian*, verb as "says".

`findRelationshipByRegex`

```
<string> text, <string> subject, <string> object <string> verb findRelationshipByRegex (<string> regex, <string> string-variable )
```

- **Extension Type:** Function

- **Description:** it returns whole text, subject, object and verb from the text stream that match with the named nodes of the Semgrep pattern

- The `findRelationshipByRegex` function takes in
 - regex: user given regular expression that match the Semgrep pattern syntax
 - string-variable : a string or the stream attribute which the text stream resides
- It returns the entities in the text. If we give group successive matches as true the result will aggregate successive words of the same entity type.

- **Examples:** `findRelationshipByRegex('={}verb >/nsubj|agent/ {}=subject >/dobj/ {}=object', "gates foundation donates $50M in support of #Ebola relief")`, returns 4 parameters. the whole text, subject as "foundation", object as "\$", verb as "donates".

`findSemgrepPattern`

<string > **text**, <string> **match**, < string > **object** < string > **verb** **findSemgrepPattern** (<string> **regex**, <string> **string-variable**)

- **Extension Type:** Function
- **Description:** it returns whole text, subject, object and verb from the text stream that match with the named nodes of the Semgrep pattern
 - The findSemgrepPattern function takes in
 - regex: user given regular expression that match the Semgrep pattern syntax
 - string-variable : a string or the stream attribute which the text stream resides
 - it returns word(s)/phrase(s) from the text stream that match with the Semgrep pattern and word(s)/relation(s) that match with each named node and each named relation defined in the regular expression.
- **Examples:** findSemgrepPattern('{lemma:die} >/.*subj|num.*/=reln {}=diedsubject', "Sierra Leone doctor dies of Ebola after failed evacuation.") returns 4 parameters. the whole text, match as "dies", reln as "nsubj", diedsubject as "doctor".
 - This will look for words with lemmatization *die* which are governors on any subject or numeric relation. The dependent is marked as the *diedsubject* and the relationship is marked as *reln*. Thus, the query will return an output stream that will out the full match of this expression, i.e the governing word with lemmatization for die. In addition it will out the named node diedsubject and the named relation reln for each match it find.

findTokensRegexPattern

< string > **text**, <string> **match**, <string> **group_1**, etc. **findTokensRegexPattern** (<string> **regex**, <string> **string-variable**)

- **Extension Type:** Function
- **Description:** it returns whole text, subject, object and verb from the text stream that match with the named nodes of the Semgrep pattern
 - The findTokensRegexPattern function takes in
 - regex: user given regular expression that match the Semgrep pattern syntax
 - string-variable : a string or the stream attribute which the text stream resides
 - it returns word(s)/phrase(s) from the text stream that match with the Semgrep pattern and word(s)/relation(s) that match with each named node and each named relation defined in the regular expression.
- **Examples:** findTokensRegexPattern('([ner:/PERSON|ORGANIZATION|LOCATION/]+) (?::[]*) [lemma:donate] ([ner: MONEY]+)', text) returns 4 parameters. the whole text, match as " Paul Allen donates \$ 9million ", group_1 as " Paul Allen", group_2 as "\$ 9million".
 - It defines three groups and the middle group is defined as a non capturing group. The first group looks for words that are entities of either PERSON, ORGANIZATON or LOCATION with one or more successive words matching same. Second group represents any number of words followed by a word with lemmatization for donate such as donates, donated, donating etc. Third looks for one or more successive entities of type MONEY.

pmmI

< double | float|long|int|string|boolean > **predict**(<string> **pathToPmmlFile**)

- **Extension Type:** Stream Processor
- **Description:** Process the input stream attributes according to the defined PMML standard model and outputs the processed results along with the input stream attributes.
 - The predict function takes in
 - pathToPmmlFile: path to the PMML model file
 - Returns the outputs defined in the output fields. The number of outputs can be varied.
- **Examples:** predict('<DAS HOME>/samples/artifacts/0301/decision-tree.pmml')
 - This model is implemented to detect network intruders. The input event stream is processed by the execution plan 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 responses along with the feature values extracted from the input event stream.

```
< double | float|long|int|string|boolean > predict(<string> pathToPmmlFile, <double|float|long|int|string|boolean> input )
```

- **Extension Type:** Stream Processor
- **Description:** Process the input stream attributes according to the defined PMML standard model and outputs the processed results.
 - The predict function takes in
 - pathToPmmlFile: path to the PMML model file
 - input: attribute of the input stream which is sent to the PMML model as values for predictions . Function does not accept any constant values as input parameters. You can have multiple input parameters according to the input stream definition.
 - Returns the processed outputs defined in the query. The number of outputs can be varied according to the query definition.
- **Examples:** predict('<DAS HOME>/samples/artifacts/0301/decision-tree.pmml', root_shell double, su_attempted double, num_root double, num_file_creations double, num_shells double, num_access_files double, num_outbound_cmds double, is_host_login double, is_guest_login double, count double, srv_count double, serror_rate double, srv_serror_rate double)
 - This model is implemented 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 responses.

Creating Siddhi Query Templates

Domain-specific execution manager is a new feature in WSO2 DAS. It lets you customize, preconfigured domain-specific execution parameter values in an user friendly dashboard. The dashboard allows you to configure these parameter values through available domain templates.

Generally, you need to edit an execution plan by writing the required Siddhi queries, if you want to monitor a specific event flow or modify a parameter value in WSO2 DAS. However, with this execution manager feature, you can focus on the query combinations by visualizing Siddhi queries with the help of the available template structures. With this feature you can identify a combination of choices to obtain the desired output stream of events.

Following topics describe the functionality of the domain-specific execution manager feature.

- Creating a template
- Configuring the execution plan
- Configuring the parameters
- Configuring the event streams
- Execution manager dashboard

Creating a template

You need to create template configurations specific to your domain in a XML file, and place them in the <DAS_HOME>/repository/conf/cep/domain-template/ directory, to use the execution manager and populate the Siddhi Queries. By default, WSO2 DAS contains a sample domain named TemperatureAnalysis. Following is an example template configuration file specific for this TemperatureAnalysis domain.

You can have one or more templates in a domain. Each template has an execution plan with a set of parameters for that execution plan which you can configure.

```
<templateDomain name="TemperatureAnalysis">
    <description>Temperature Analysis Description</description>
    <templates>
        <template name="Maximum Temperature">
            <description>To check maximum room temperature of all rooms</description>
            <executionPlan><![CDATA[
                /* Enter a unique ExecutionPlan */
            
```

```

        @Plan:name('testPlan')

        /* Enter a unique description for ExecutionPlan */
        -- @Plan:description('ExecutionPlan')

        /* define streams and write query here ... */

        @Import('inStream:1.0.0')
        define stream inStream (meta_temperature double,
meta_roomNumber int);

        @Export('outStream:1.0.0')
        define stream outStream (meta_temperature double,
meta_roomNumber int);

        from inStream[meta_temperature > $maxVal]
        select meta_temperature,meta_roomNumber
        insert into outStream;
    ]]></executionPlan>

<parameters>
    <parameter name="maxVal" type="int">
        <displayName>Maximum Temperature</displayName>
        <description>Maximum room temperature threshold</description>
        <defaultValue>75</defaultValue>
    </parameter>
</parameters>
</template>
</templates>
<streams>
    <stream>
        {
            "streamId": "inStream:1.0.0",
            "name": "inStream",
            "version": "1.0.0",
            "nickName": "",
            "description": "",
            "metaData": [
                {
                    "name": "temperature",
                    "type": "DOUBLE"
                },
                {
                    "name": "roomNumber",
                    "type": "INT"
                }
            ],
            "correlationData": [],
            "payloadData": []
        }
    </stream>
    <stream>
        {
            "streamId": "outStream:1.0.0",
            "name": "outStream",
            "version": "1.0.0",
            "nickName": "",
            "description": "",
            "metaData": [
                {

```

```
        "name": "temperature",
        "type": "DOUBLE"
    } ,
    {
        "name": "roomNumber",
        "type": "INT"
    }
],
"correlationData": [ ],
"payloadData": [ ]
}
```

```

        </stream>
</streams>
</templateDomain>
```

Configuring the execution plan

The execution plan is deployed in a template in WSO2 DAS, once you save a configuration for the first time. You can modify the name specified in the execution plan, and re-deploy it by adding a configuration name. For more information on execution plans, see [Creating a Standalone Execution Plan](#).

Name the configurable parameters of the Siddhi query in the execution plan by using the “\$” character. For example, in the sample query below, \$timeInterval and \$maxVal are the configurable parameters which should be defined within the `<parameters>` element of the template.

```

from inStream#window.time($timeInterval)
select avg(temperature) as temperature, roomNumber
group by roomNumber
having temperature >= $maxVal
insert into outStream;
```

Configuring the parameters

A template can have one or more parameters. You can set a display name and a description to show in the user interface. Also, you can set a default value if required. If the input values need to be limited to a defined set of values, define the options as comma-separated values. These values are shown in a drop down list in the user interface. An example configuration of a set of parameters is shown below.

```

<parameters>
    <parameter name="maxVal" type="int">
        <displayName>Maximum Temperature</displayName>
        <description>Maximum Temperature which needs to be checked</description>
        <defaultValue>75</defaultValue>
        <options>75, 80, 85</options>
    </parameter>
    <parameter name="timeInterval" type="time">
        <displayName>Time Interval</displayName>
        <description>Time can be defined such 5 sec, 1 min and etc</description>
        <defaultValue>1 min</defaultValue>
    </parameter>
</parameters>
```

Configuring the event streams

You can have one or more event streams in a domain template. Define the required event streams within the `<streams>` element of the domain template. The input and output event streams that are defined in the template are deployed in WSO2 DAS, once you save a configuration for the first time. For information on defining event streams, see [Event Streams](#). An example configuration of event streams is shown below.

```

<streams>
  <stream>
    {
      "streamId": "inStream:1.0.0",
      "name": "inStream",
      "version": "1.0.0",
      "nickName": "",
      "description": "",
      "metaData": [
        {
          "name": "temperature",
          "type": "DOUBLE"
        },
        {
          "name": "roomNumber",
          "type": "INT"
        }
      ],
      "correlationData": [],
      "payloadData": []
    }
  </stream>
  <stream>
    {
      "streamId": "outStream:1.0.0",
      "name": "outStream",
      "version": "1.0.0",
      "nickName": "",
      "description": "",
      "metaData": [
        {
          "name": "temperature",
          "type": "DOUBLE"
        },
        {
          "name": "roomNumber",
          "type": "INT"
        }
      ],
      "correlationData": [],
      "payloadData": []
    }
  </stream>
</streams>

```

Execution manager dashboard

Use the execution manager dashboard to do the configurations.

Follow the steps below to use the execution manager dashboard.

1. Log in to the management console using the following URL: https://<DAS_HOST>:<DAS_PORT>/carbon
2. Click **Main**, and then click **Execution Manager** under the **Manage** menu. You view the dashboard home page with the available domains as shown below.



Domains

TemperatureAnalysis

TemperatureAnalysis

3. Click on the domain, which you need to configure.
4. Click **Add Configuration** to add a new configuration to the selected domain as shown below.

Configurations

Execution Manager / TemperatureAnalysis

Add Configuration

No Configurations to be listed

5. Enter a desired **Configuration Name**, **Description**, and values for the **Parameter Configurations** for the template configuration information as shown below.

Edit Configurations

Execution Manager / TemperatureAnalysis / Edit Configurations

Template	<input style="border: none; padding: 2px 5px; margin-right: 5px;" type="button" value="Template1"/> To check maximum room temperature of all rooms
Configuration Name	<input type="text" value="Temperature_Analysis_Configuration"/>
Description	<input type="text" value="This is a temperature analysis configuration."/>

PARAMETER CONFIGURATIONS

Maximum Temperature	<input type="text" value="55"/>
	Maximum Temperature which needs to be checked

Add Configuration

- Click **Add Configuration**. You view a pop-up message on successful addition of the configuration as shown below.



- Close the pop-up message. You view the new configuration in the list of tall available configurations of the selected domain as shown below. You can edit or delete it using the corresponding options provided.

Configurations

Execution Manager / TemperatureAnalysis

Add Configuration

Configuration Name	Description	Template Type	Actions	
MainConfiguration	This is main default configuration	Template3	Delete	Edit

An execution plan with the configured parameters gets deployed, once you save the configuration. Follow the steps below to view this new execution plan.

- Log in to the management console using the following URL, if you are not already logged in: https://<DAS_HOST>:<DAS_PORT>/carbon/
- Click **Main**, and then click **Execution Plans** under the **Event Processor** menu. You view the new execution plan added to the list of available execution plans as shown below.

Home > Manage > Event Processor > Execution Plans Help

Available Execution Plans

[Add Execution Plan](#)

1 Active Execution Plans. 0 Inactive Execution Plans (All)

Execution Plan Name	Description	Actions
TemperatureAnalysis-MainConfiguration		Enable statistics Enable Tracing Delete Edit

- Click on the name of the execution plan. The parameters you entered are applied to the execution plan as shown below.

Event Processor Details

Execution Plan

```

2      /* Enter a unique ExecutionPlan */
3      @Plan:name('TemperatureAnalysis-MainConfiguration')
4
5      /* Enter a unique description for ExecutionPlan */
6      -- @Plan:description('ExecutionPlan')
7
8      /* define streams and write query here ... */
9
10     @Import('inStream:1.0.0')
11     define stream inStream (temperature double, roomNumber int);
12
13     @Export('outStream:1.0.0')
14     define stream outStream (temperature double, roomNumber int);
15
16     from inStream#window.time(1 min)
17     select avg(temperature) as temperature, roomNumber
18     group by roomNumber
19     having temperature >= 91
20     insert into outStream;
21

```

The required event streams defined in the execution plan also gets deployed, once you save the configuration. Follow the steps below to view the event streams.

11. Log in to the management console using the following URL, if you are not already logged in: https://<DAS_HOST>:<DAS_PORT>/carbon/
12. Click **Main**, and then click **Execution Streams** under the **Event Processor** menu. You view the deployed event streams in the list of available event streams as shown below.

Available Event Streams

Add Event Stream

2 Event streams available

Event Stream Id	Event Stream Description	Actions
inStream:1.0.0		Delete Edit
outStream:1.0.0		Delete Edit

Predictive Analytics Using WSO2 ML

Following sections describe how you can perform predictive analytics using WSO2 Machine Learner (ML).

- Configuring WSO2 ML with WSO2 DAS
- Creating Machine Learning Models
- Using a ML Model Within WSO2 CEP
- Using a ML Model Within WSO2 ESB

Configuring WSO2 ML with WSO2 DAS

For information on configuring WSO2 ML with WSO2 DAS, go to [Integration with WSO2 Data Analytics Server](#).

Creating Machine Learning Models

For information on creating ML models using WSO2 Machine Learner (ML), go to [WSO2 ML User Guide](#).

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](#)

Analytics Dashboard

Analytics dashboard application is the data visualization component of WSO2 Data Analytics Server. You can create dashboards using summarized analytics tables and event streams. A dashboard is a container for gadgets/widgets, and its pages have a structured layout with predefined grids, to which you add the dashboard components. The analytics dashboard serves the following purposes.



Note that this dashboard is not supported with IBM JDK.



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](#).

- [Adding a Dashboard](#)
- [Adding Pages to a Dashboard](#)
- [Adding a Layout to a Page](#)
- [Adding Gadgets to a Dashboard](#)
- [Adding Widgets to a Dashboard](#)

Adding a Dashboard

You can create new dashboards in the analytics dashboard as explained in the following sections.

- [Creating a dashboard](#)

- Setting permissions of a dashboard
- Viewing the preview of a dashboard
- Exporting a dashboard

Creating a dashboard

Follow the steps below to create a dashboard.

1. Access the analytics dashboard application using the following URL:

https://<HOST_NAME>:<PORT>/portal/dashboards. (E.g. <https://localhost:9443/portal/dashboards>). You are presented with a login screen like below.



Login

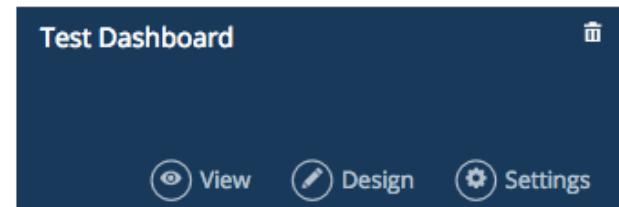
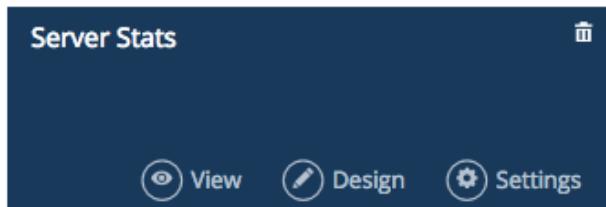
admin

Login

2. Log in to the dashboard using Carbon login credentials. You are presented with a list of dashboards you've already created as shown below.

DASHBOARDS

* View, Modify and Delete dashboards



3. Click the following **CREATE DASHBOARD** button in the top navigational bar to create a new dashboard.



4. Enter a **Title** and a **Description** for the new dashboard as shown below, and click **Next**.

CREATE A DASHBOARD

Name of your Dashboard

WSO2 DAS Test Dashboard

URL

wso2-das-test-dashboard

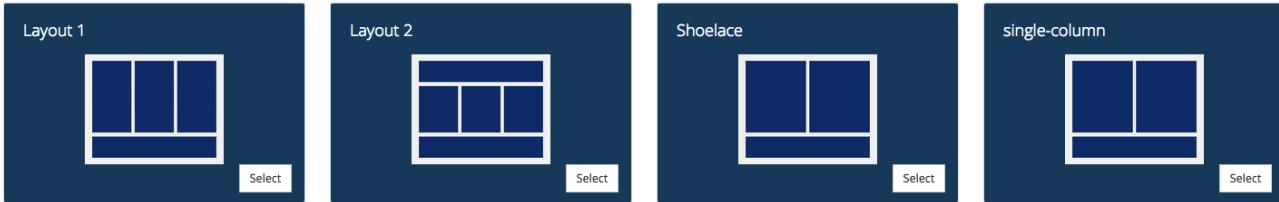
Description

This is a test dashboard in WSO2 DAS.

Next

5. Select a layout to place its components as shown below.

PLEASE SELECT A LAYOUT



6. Click **Select**. You view a layout editor with the chosen layout blocks marked using dashed lines as shown below. Now the dashboard is persisted to the disk.

The image shows the WSO2 Data Analytics Server dashboard design interface. The top navigation bar includes "DASHBOARD DESIGN", "ADD PAGE", "PREVIEW", "EXPORT", "SETTINGS", and "PAGES". On the left, there are icons for "Welcome" and "Dashboards". The main area displays a layout editor with a 4x3 grid of dashed lines, indicating the selected layout blocks. A large orange square icon is positioned at the bottom center of the grid.

7. Click the following icon in the top menu.
8. Click **Dashboards** to view the new dashboard added to the list of all available dashboards as shown below.



You can edit, view or delete the dashboards using the corresponding options provided.

DASHBOARDS

* View, Modify and Delete dashboards

The screenshot shows a grid of three dashboard cards. The first card is 'Server Stats' with icons for View, Design, and Settings. The second card is 'Test Dashboard' with the same icons. The third card is 'WSO2 DAS Test Dashboard', which contains the text 'This is a test dashboard in WSO2 DAS.' and also has icons for View, Design, and Settings. A red box highlights the 'WSO2 DAS Test Dashboard' card.

Setting permissions of a dashboard

Follow the steps below to set permissions of a dashboard.

1. Log in to the analytics dashboard using Carbon login credentials, if you are not already logged in.
2. Click **Settings** of the corresponding dashboard in which you want to set permissions.
3. Enter the user roles to which you would like to grant respective permissions to view and edit for **Permissions** as shown below.

i By default, the user role Internal/everyone is set for **Editors**.

The screenshot shows two sections for setting permissions. The 'Viewers' section has a 'Roles' input field containing 'admin ✕' and 'Internal/everyone ✕'. The 'Editors' section has a 'Roles' input field containing 'Internal/everyone ✕' and 'admin ✕'.

Viewing the preview of a dashboard

Follow the steps below to view the preview of a dashboard.

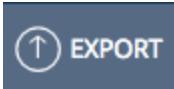
1. Log in to the analytics dashboard using Carbon login credentials, if you are not already logged in.
2. Click **View** of the corresponding dashboard in which you want to view.
3. You view the preview of the selected dashboard with all its pages in different tabs as shown below.

The screenshot shows a dashboard preview with a top navigation bar for 'WSO2 DAS Test Dashboard' with tabs 'Home' (selected), 'Page 0', and 'Page 1', and a user 'admin'. Below the tabs are two content panels: 'USA Business Revenue' (empty) and a calendar for July 2015. The calendar shows dates from 28 to 31 July, with the 29th and 30th highlighted.

Exporting a dashboard

Follow the steps below to export a dashboard configuration as a JSON file.

1. Log in to the analytics dashboard using Carbon login credentials, if you are not already logged in.
2. Click **Design** of the corresponding dashboard which you want to export.
3. Click the following button in the top menu bar.



4. You view the dashboard configuration exported to JSON format in a new tab of your web browser as shown below.

Adding Pages to a Dashboard

Dashboards created in the analytics dashboard of WSO2 DAS has one or more pages. By default, a dashboard you create has a page named **My Dashboard** set as the landing page. You can add more pages to a dashboard. Follow the steps below to add a page to a dashboard.

1. Access the analytics dashboard application using the following URL:
`https://<HOST_NAME>:<PORT>/portal/dashboards.`
 2. Log in to the dashboard using Carbon login credentials.
 3. Click the corresponding **Edit** button of the dashboard to which you want to add a page as shown below.

WSO2 DAS Test Dashboard

4. Click the following button in the dashboard editor screen to add a new page.



5. Click **Select** of the layout which you want to add to the page out of the existing layouts. For instructions on adding a layout to a page, see [Adding a Layout to a Page](#).
 6. Click **Pages**, and then click **Page 0** to view the new page you added in the dashboard editor as shown below.

PAGES
WELCOME
PAGE 0
PAGE 1
PAGE 2

 Click the following icon to delete the selected page:

7. Click the properties icon, to change the properties of the selected page as shown below.

The screenshot shows the 'PAGES' section of the WSO2 Data Analytics Server. A page titled 'Page 2' is selected. The 'Properties' tab is active, highlighted with a red border. Inside the properties panel, there are fields for 'Title' (containing 'Page 2') and 'URL' (containing 'page2'). Below these fields is a checked checkbox labeled 'Use as Landing'.

Adding a Layout to a Page

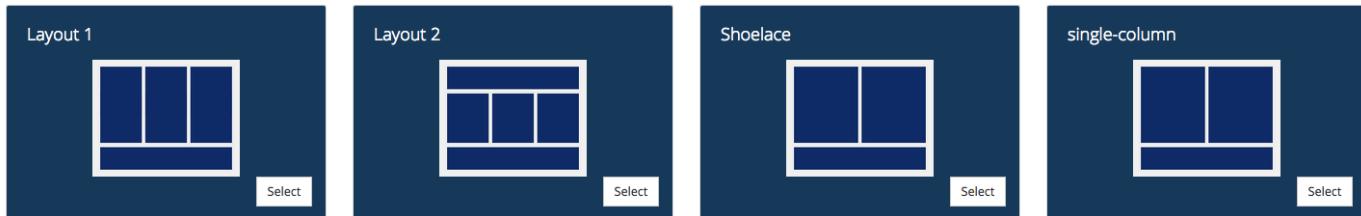
Layout is the structure which defines how the dashboard components (gadgets and widgets) are arranged within a page. You can define a layout to the home page, which is named **My Dashboard** by default when [creating a new dashboard](#), or to a new page when you are [creating it](#). You can select it from the layouts that are available by default, or by adding a custom layout as described below.

- Selecting a default layout
- Creating a custom layout

Selecting a default layout

Following are the layouts available by default in the analytics dashboard of WSO2 DAS.

PLEASE SELECT A LAYOUT



Creating a custom layout

Use the sample layout configuration structures in the `<Das_Home>/repository/deployment/server/jagger/yapps/portal/store/carbon.super/layout/` directory to create a custom layout. Add the following resources and create the configuration parent directory of your custom layout.

1. Add the layout format as shown in the example (`index.hbs` file) below.

```
<div class="row">
    <div id="a" class="col-md-3 ues-component-box"></div>
    <div id="b" class="col-md-6 ues-component-box"></div>
    <div id="c" class="col-md-3 ues-component-box"></div>
</div>
<div class="row">
    <div id="d" class="col-md-6 ues-component-box"></div>
    <div id="e" class="col-md-6 ues-component-box"></div>
</div>
<div class="row">
    <div id="f" class="col-md-4 ues-component-box"></div>
    <div id="g" class="col-md-4 ues-component-box"></div>
    <div id="h" class="col-md-4 ues-component-box"></div>
</div>
```

- Add the layout definition as shown in the example (layout.json file) below.

```
{
    "id": "my-layout",
    "title": "My Layout",
    "description": "This is a sample grid",
    "thumbnail": "local://store/layout/my-layout/index.png",
    "url": "local://store/layout/my-layout/index.hbs"
}
```

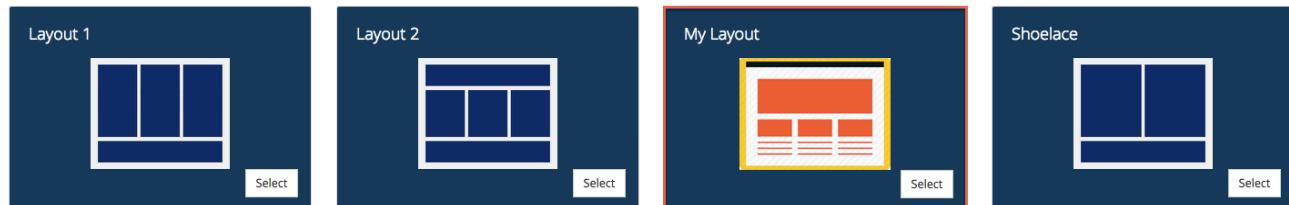
- Add a thumbnail image (index.jpg) to display in the list of layouts in the analytics dashboard as the below example.



- Refresh the analytics dashboard application.

Now, you view the new layout you created added to the list of available layouts as shown in the below example.

PLEASE SELECT A LAYOUT



You can add this custom layout to a new dashboard or to a new page.

Adding Gadgets to a Dashboard

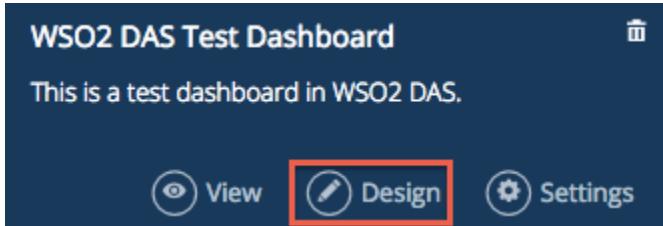
A gadget is a component of a dashboard. You can place gadgets to the pre-defined grids of the layout of a page as described below.

- Adding a default gadget
- Adding a gadget using the gadget generation wizard
- Editing a gadget
- Creating a custom gadget

Adding a default gadget

Follow the steps below to add a gadget which is available by default in the analytics dashboard of WSO2 DAS.

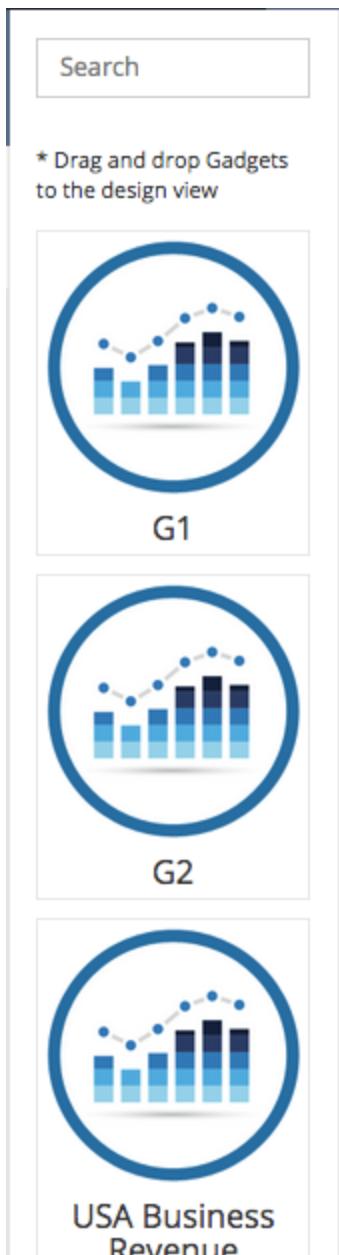
1. Access the analytics dashboard application using the following URL:
https://<HOST_NAME>:<PORT>/portal/dashboards.
2. Log in to the dashboard using Carbon login credentials.
3. Click the corresponding **Design** button of the dashboard to which you want to add a gadget as shown below.



4. Click the following gadget browser icon in the side menu bar.



You view all available gadgets as shown below.



- Drag the gadget out which you want to add, and drop it to place it in the preferred grid of the selected layout in the dashboard editor as shown below.

The screenshot shows a dashboard editor layout grid. The top bar says "Welcome". The layout consists of a 2x2 grid of cells. The bottom-right cell contains a gadget titled "USA Business Revenue" with a red border around its title. The other cells are empty.

Adding a gadget using the gadget generation wizard

Gadget generation wizard is a UI based fast and easy way to generate a gadget without writing any code. It guides you through multiple steps to customize the parameters of the gadget accordingly in each step. The gadget generation wizard has the following two steps.

- Selecting a datasource
- Configuring the chart

Selecting the data source

Follow the steps below to select a data source to get data for the gadget.

 You need to create a UI event publisher associated with a realtime event stream or have a table created in the Data Access Layer, to select it as the source of data to generate the gadget.

1. Log in to the dashboard using Carbon login credentials, if you are not already logged in.
2. Click the following **CREATE GADGET** icon in the top menu bar.



3. Select the input data source as shown below. It can be either an analytics table or an event stream.

CREATE A GADGET

Select Table/Event Stream

Configure Chart

Analytics Table/Event Stream

USAGESTREAM [batch]

Preview Data

4. Click **Preview Data** to view a preview of the data source as shown below.

CREATE A GADGET

Select Datasource

Configure Chart

Datasource

JMX_AGENT_TOOLBOX [batch]

Preview Data

heap_mem_init	non_heap_mem_used	peakThreadCount	heap_mem_max	heap_mem_used	meta_host	non_heap_mem_max	threadCount
268435456	135533120	369	954728448	231868104	localhost:11111	318767104	365
268435456	135727744	369	954728448	261247784	localhost:11111	318767104	367
268435456	135739968	370	954728448	266242384	localhost:11111	318767104	368
268435456	135743840	372	954728448	271700880	localhost:11111	318767104	370
268435456	135743840	372	954728448	276780472	localhost:11111	318767104	370
268435456	135744360	373	954728448	280691624	localhost:11111	318767104	371
268435456	135755640	374	954728448	286849968	localhost:11111	318767104	372
268435456	135775288	375	954728448	290696928	localhost:11111	318767104	373
268435456	135776808	376	954728448	299832992	localhost:11111	318767104	374
268435456	135781032	377	954728448	305090920	localhost:11111	318767104	375

Previous

Next

5. Click **Next**.

Configuring the chart

Follow the steps below to configure the chart.

1. Select **Chart Type** and enter the preferred x, y axis and additional parameters based on the selected chart type as shown below.

CREATE A GADGET

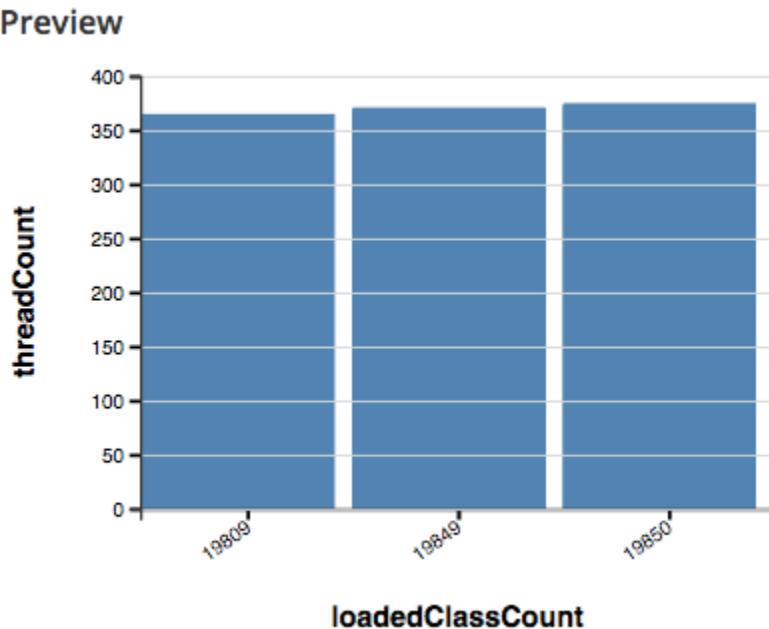
Select Datasource **Configure Chart**

Configuration

Chart Type	Bar
Title	Bar Chart
X-Axis	loadedClassCount
Y-Axis	threadCount

Preview

2. Click **Preview** to view a preview of the chart as shown below.



3. Once all the customizations are done, click **Add to Gadget Store** to generate a gadget with the information you provided.
4. Click the corresponding **Design** button of the dashboard to which you want to add a gadget as shown below.

WSO2 DAS Test Dashboard

This is a test dashboard in WSO2 DAS.

(View) (Design) (Settings)

5. Click the following gadget browser icon in the side menu bar.



You view the new gadget listed in the gadget browser as shown below.

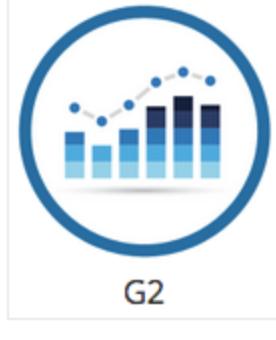
* Drag and drop Gadgets
to the design view



Bar Chart



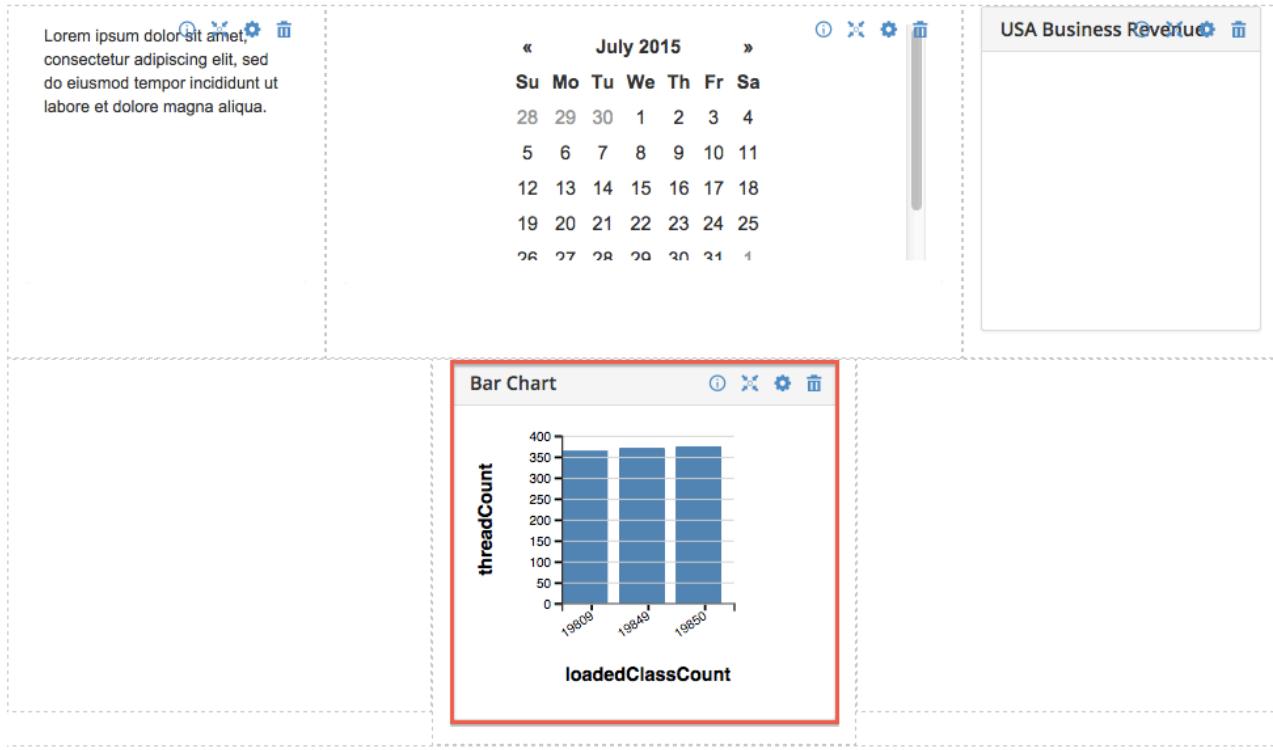
G1



G2

6. 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.

Welcome



Editing a gadget

You can use the following options provided in the dashboard editor to edit a gadget as described below.



- Click the following icon to view information about the gadget in a tooltip:
- Click the following icon to move the gadget (drag-and-drop) from one grid to another in the layout.
- Click the following icon to view the settings of the gadget. You can change the default settings of the gadget as preferred as shown in the below example.

Bar Chart

Options

Data Source

Update Interval (s)

- Click the following icon to remove the gadget from the layout.



Creating a custom gadget

Use the sample gadget configuration structures in the <DAS_HOME>/repository/deployment/server/jettyapps/portal/store/carbon.super/gadget/ directory to create a custom gadget. Add the following resources and create the configuration parent directory of your custom gadget.

- Add the gadget structure as shown in the example (`index.xml` file) below.

```

<?xml version="1.0" encoding="UTF-8" ?>
<Module>
    <ModulePrefs title="USA Mobile Stats" height="250" description="USA Mobile
Stats">
        <Require feature="pubsub-2"/>
        <Require feature="dynamic-height"/>
    </ModulePrefs>
    <Content type="html">
        <![CDATA[
            <head>
                <style type="text/css">
                    .log {
                        width: 400px;
                        height: 400px;
                        background-color: #415b76;
                        color: #fff;
                    }
                </style>
                <script language="javascript" type="text/javascript"
src="/portal/js/jquery-1.10.2.js"></script>
                <script>
                    gadgets.HubSettings.onConnect = function() {
                        var id = 0;
                        setInterval(function() {
                            $('.log').append('<div>publishing message from g1, id : '
+ (++id) + '</div>');
                            gadgets.Hub.publish('select', {
                                msg : 'A message from g1',
                                id: id
                            });
                        }, 5000);
                    };
                </script>
            <head>
            <body>
                <div class="log"></div>
            </body>
        ]]>
    </Content>
</Module>

```

2. Add the gadget definition as shown in the example `gadget.json` file) below.

```
{
    "id": "USA Mobile Stats",
    "title": "USA Mobile Stats",
    "type": "gadget",
    "thumbnail": "local://store/gadget/my-gadget/index.png",
    "data": {
        "url": "local://store/gadget/my-gadget/index.xml"
    },
    "notify": {
        "select": {
            "type": "address",
            "description": "This generates mobile stats."
        },
        "cancel": {
            "type": "boolean",
            "description": "This generates disconnected mobile stats."
        }
    }
}
```

3. Add a thumbnail image (index.png) to display in the list of gadgets in the analytics dashboard as the below example



4. Refresh the analytics dashboard application.

Now, you view the new gadget you created added to the list of available gadgets as shown in the below example.

* Drag and drop Gadgets to the design view



You can add this custom gadget to a layout.

Adding Widgets to a Dashboard

A widget is a component of a dashboard. You can place widgets to the pre-defined grids of the layout of a page as described below.

- Adding a default widget
- Editing a widget
- Creating a custom widget

Adding a default widget

Follow the steps below to add a widget which is available by default in the analytics dashboard of WSO2 DAS.

1. Access the analytics dashboard application using the following URL:
https://<HOST_NAME>:<PORT>/portal/dashboards.
2. Log in to the dashboard using Carbon login credentials.
3. Click the corresponding **Design** button of the dashboard to which you want to add a gadget as shown below.

WSO2 DAS Test Dashboard

This is a test dashboard in WSO2 DAS.

(View) (Design) (Settings)

4. Click the following widget browser icon in the side menu bar.



You view all available widgets as shown below.

Search

* Drag and drop Widgets to the design view



Date Picker



Text Box

5. Drag the widget out which you want to add, and drop it to place it in the preferred grid of the selected layout in the dashboard editor as shown below.

Welcome

July 2015

«	July 2015	»				
Su	Mo	Tu	We	Th	Fr	Sa
28	29	30	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	1

USA Business Revenue

Editing a widget

You can use the following options provided in the dashboard editor to edit a widget as described below.



- Click the following icon to view information about the widget in a tooltip.
- Click the following icon to move the widget (drag-and-drop) from one grid to another in the layout.
- Click the following icon to view the settings of the widget.

You can change the default settings of the widget as preferred as shown in the below example.



- Click the following icon to remove the widget from the layout.

Creating a custom widget

Use the sample widget configuration structures in the `<Das_Home>/repository/deployment/server/jettyapps/portal/store/carbon.super/widget/` directory to create a custom gadget. Add the following resources and create the configuration parent directory of your custom gadget.

1. Add the widget structure as shown in the example (`index.xml` file) below.

```

<?xml version="1.0" encoding="UTF-8" ?>
<Module>
    <ModulePrefs title="Temp Search" description="Temp Search">
        <Require feature="dynamic-height"/>
    </ModulePrefs>
    <UserPref name="content"
        display_name="Content"
        default_value="Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua."/>
    <Content type="html">
        <![CDATA[
            <head>
                <meta charset="utf-8">
                <title>Text Box</title>
                <link rel="stylesheet" href="/portal/css/bootstrap.min.css"
type="text/css"/>
                <style>
                    .textbox {
                        padding-left: 0;
                        padding-right: 0;
                        padding-top: 0;
                    }
                </style>
                <script src="/portal/js/jquery-1.10.2.js"></script>
                <script src="/portal/js/bootstrap.min.js"></script>
                <script>
                    gadgets.util.registerOnLoadHandler(function() {
                        var prefs = new gadgets.Prefs();
                        var content = prefs.getString('content');
                        $('.textbox').html(gadgets.util.unescapeString(content));
                    });
                </script>
            </head>
            <body>
                <div class="container-fluid">
                    <div class="row">
                        <div class="col-md-12 textbox"></div>
                    </div>
                </div>
            </body>
        ]]>
    </Content>
</Module>

```

2. Add the widget definition as shown in the example (widget.json file) below.

```
{  
    "id": "Temp Search",  
    "title": "Temp Search",  
    "type": "gadget",  
    "thumbnail": "store://widget/my-widget/index.png",  
    "data": {  
        "url": "store://widget/my-widget/index.xml"  
    },  
    "options": {  
        "content": {  
            "type": "text"  
        }  
    },  
    "styles": {  
        "borders": false,  
        "title": false  
    }  
}
```

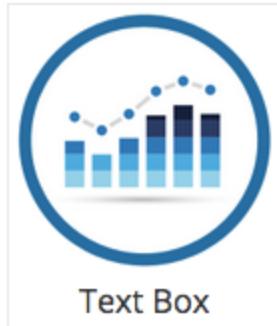
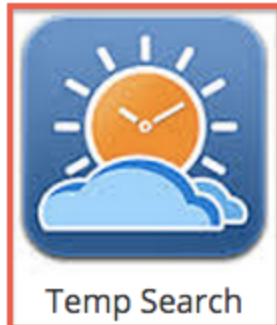
3. Add a thumbnail image (`index.png`) to display in the list of gadgets in the analytics dashboard as the below example .



4. Refresh the analytics dashboard application.

Now, you view the new widget you created added to the list of available widgets as shown in the below example.

* Drag and drop Widgets to the design view



You can add this custom widget to a layout.

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

Enter Event Publisher Details

Event Publisher Name* <input type="text" value="testEventStream:1.0.0"/> From Event Source* <input type="text" value="sensor_id int"/> Stream Attributes <div style="border: 1px solid #ccc; height: 100px; margin-top: 10px;"></div>	<small>Enter a unique name to identify Event Publisher</small> <small>The stream of events that need to be published</small>
To Output Event Adapter Type* <input type="text" value="websocket"/> <small>Select the type of Adapter to publish events</small> Static Adapter Properties Websocket Server URL* <input type="text" value="ws://localhost:9099"/> <small>URL of the web socket server which the events to be published; e.g. "ws://localhost:9099".</small> Mapping Configuration Message Format* <input type="text" value="text"/> <small>Select the output message format</small> + Advanced	
<input type="button" value="Add Event Publisher"/>	

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	Output event adapters contain three types of adapter properties in their configuration as explained below.
	<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. http://localhost:8000/endpoint/{{endpointId}} Global Adapter Properties : These properties come from the <PRODUCT_HOME>/repository/conf/output-event-adapters.xml file. They are common for all adapters on its kind which was defined during the event publisher creation.

Creating event publishers

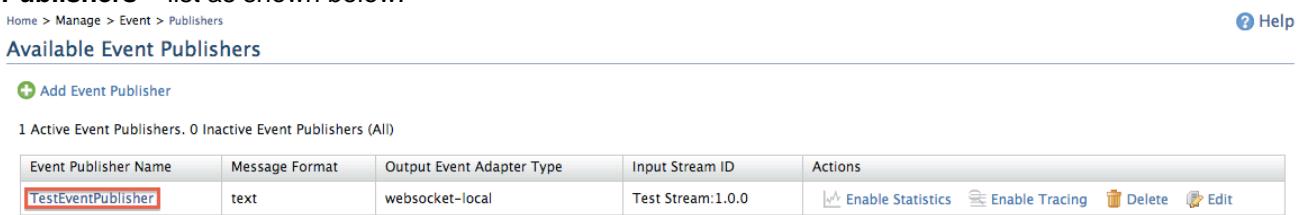
You can create event publishers either [using the management console](#) or [using a configuration file](#) as explained below.

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.

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.



The screenshot shows the 'Available Event Publishers' list. At the top, there is a navigation bar with 'Home > Manage > Event > Publishers'. On the right, there is a 'Help' link. Below the navigation, there is a header with 'Available Event Publishers' and a 'Add Event Publisher' button. A note says '1 Active Event Publishers. 0 Inactive Event Publishers (All)'. The main table has columns: 'Event Publisher Name', 'Message Format', 'Output Event Adapter Type', 'Input Stream ID', and 'Actions'. One row is visible, showing 'TestEventPublisher' in the first column, 'text' in the second, 'websocket-local' in the third, 'Test Stream:1.0.0' in the fourth, and 'Enable Statistics', 'Enable Tracing', 'Delete', and 'Edit' in the fifth column. The 'Event Publisher Name' column for 'TestEventPublisher' is highlighted with a red border.

Creating publishers using a configuration file

Follow the steps below to create an event publisher using a configuration file.

1. Create an XML file with the following event publisher configurations. An event publisher implementation must

start with `<eventPublisher>` as the root element.

⚠ 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 `<PRODUCT_HOME>/repository/deployment/server/eventpublishers/` directory. Since hot deployment is supported in the product, you can simply add/remove event publisher configuration files to deploy/undeploy event publishers to/from the server.

⚠ First define the stream to which the publisher is publishing data from to activate the publisher. When receiving WSO2Events, the outgoing stream definition that you select in the advanced input mappings must also be defined, to activate the event publisher. When you click **Inactive Event Publishers** in the **Available Event Publishers** screen, if an event publisher is in the inactive state due to some issue in the configurations, you view a short message specifying the reason why the event publisher is inactive as shown below. A similar message is also printed on the CLI.

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

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

The screenshot shows the 'Create a New Event Publisher' dialog with the following configuration:

- Event Publisher Name:** CassandraOutputEventAdapter
- From:**
 - Event Source:** Test Stream:1.0.0
 - Stream Attributes:** sensor id int
- To:**
 - Output Event Adapter Type:** cassandra
- Static Adapter Properties (highlighted with a red border):**
 - Hosts:** localhost
 - Port:** 9160
 - User Name:** admin
 - Password:** ****
 - Keyspace Name:** CEP_KS
 - Column Family Name:** CF_Transactions
 - Strategy Class:** SimpleStrategy
 - Replication Factor:** 3
 - Indexed Columns:** group_id
- Mapping Configuration:**
 - Message Format:** map
- Advanced:**

i 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="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
S7L7ZzgYeWx7D1Gbff5X3pBN2Gh9yV0BHP1E93QtFqR7uTWi141Tr7V7ZwScwNqJbiNoV+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	any 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 in WSO2 CEP Documentation.

- 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="maxThread">100</property>
    <property key="keepAliveTimeInMillis">20000</property>
    <property key="jobQueueSize">10000</property>
</adapterConfig>
```



In gmail account security settings you may have to enable "Allow less secure apps" option in order to connect account to WSO2 products.

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	
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	
Message Format*	text ⑦ Select the output message format
Advanced	
<input type="button" value="Add Event Publisher"/>	

i 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="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	email.type	text/plain

Related samples

For more information on `email` event publisher type, see the following sample in WSO2 CEP Documentation.

- [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 ② Enter a unique name to identify Event Publisher								
From									
Event Source*	Test Stream:1.0.0 ② The stream of events that need to be published								
sensor id int									
Stream Attributes									
To									
Output Event Adapter Type*	http ② Select the type of Adapter to publish events								
Static Adapter Properties <table border="1"> <tr> <td>Proxy Host</td> <td>localhost ② The proxy server host</td> </tr> <tr> <td>Proxy Port</td> <td>8080 ② The proxy server port</td> </tr> <tr> <td>HTTP Client Method*</td> <td>HttpPost ②</td> </tr> </table>		Proxy Host	localhost ② The proxy server host	Proxy Port	8080 ② The proxy server port	HTTP Client Method*	HttpPost ②		
Proxy Host	localhost ② The proxy server host								
Proxy Port	8080 ② The proxy server port								
HTTP Client Method*	HttpPost ②								
Dynamic Adapter Properties <table border="1"> <tr> <td>URL*</td> <td>http://localhost:8080 ② The target HTTP/HTTPS URL, e.g. "http://yourhost:8080/service"</td> </tr> <tr> <td>Username</td> <td>admin ② HTTP BasicAuth username</td> </tr> <tr> <td>Password</td> <td>***** ② HTTP BasicAuth password</td> </tr> <tr> <td>Headers</td> <td>header1: value1, header2: value2 ② Custom HTTP headers, e.g. "header1: value1, header2: value2"</td> </tr> </table>		URL*	http://localhost:8080 ② The target HTTP/HTTPS URL, e.g. "http://yourhost:8080/service"	Username	admin ② HTTP BasicAuth username	Password	***** ② HTTP BasicAuth password	Headers	header1: value1, header2: value2 ② Custom HTTP headers, e.g. "header1: value1, header2: value2"
URL*	http://localhost:8080 ② The target HTTP/HTTPS URL, e.g. "http://yourhost:8080/service"								
Username	admin ② HTTP BasicAuth username								
Password	***** ② HTTP BasicAuth password								
Headers	header1: value1, header2: value2 ② Custom HTTP headers, e.g. "header1: value1, header2: value2"								
Mapping Configuration									
Message Format*	text ② Select the output message format								
+ Advanced									
<input type="button" value="Add Event Publisher"/>									

- i** 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="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">kuv2MubUUveMyv6GeHrXr9i159ajJIqUI4eoYHcgGKf/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	http.proxy.host	yourhost
Proxy Port	The proxy server port	http.proxy.port	8080
HTTP Client Method	The standard HTTP client method	http.client.method	HttpPost

Dynamic adapter properties

Adapter Property	Description	Configuration file property	Example
URL	The target HTTP/HTTPS URL	http.url	http://yourhost:8080/service or https://yourhost:8080/service
Username	HTTP BasicAuth username	http.username	admin
Password	HTTP BasicAuth password	http.password	admin
Headers	Custom HTTP headers	http.headers	header1: value1, header2: value2

Related samples

For more information on `http` event publisher type, see the following sample in WSO2 CEP Documentation.

- 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/con`

f/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 ⓘ Enter a unique name to identify Event Publisher
From	dashboardTest:1.0.0 ⓘ The stream of events that need to be published
Event Source*	x int, y string, z string
Stream Attributes	
To	jms ⓘ Select the type of Adapter to publish events
Static Adapter Properties	
JNDI Initial Context Factory Class*	org.apache.activemq.jndi.ActiveMQInitialContextFactory ⓘ JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.
JNDI Provider URL*	tcp://localhost:61616 ⓘ URL of the JNDI provider.
Username	jms-user
Password	*****
Connection Factory JNDI Name*	TopicConnectionFactory ⓘ The JNDI name of the connection factory.
Destination Type*	topic ⓘ Type of the destination.
Destination*	topicMap allow ⓘ Concurrent publishers can yield high throughput, but may result in out-of-order message delivery.
Dynamic Adapter Properties	
Header	ⓘ Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)
Mapping Configuration	
Message Format*	text ⓘ Select the output message format
Advanced	
<input type="button" value="Add Event Publisher"/> <input type="button" value="Test Event Publisher"/>	

- i** 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
qZK7dkKTlBBFNmB2czfSiJrUz9jCYxFXSUquCfqFs8UKXx3976sjmM+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 java.naming.spi.InitialContextFactory interface	java.naming.factory.initial
JNDI Provider URL	URL of the JNDI provider	java.naming.provider.url
Username	Valid username for the JMS connection	transport.jms.UserName
Password	Valid password for the JMS connection	transport.jms.Password
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
Destination	The topic or queue to which WSO2 CEP sends messages by publishing.	transport.jms.Destination

Concurrent Publishers	If concurrent publishers are allowed to publish events to a JMS broker using multiple threads.	transport.jms.ConcurrentPublishers
------------------------------	--	------------------------------------

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 in WSO2 CEP Documentation.

- [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.

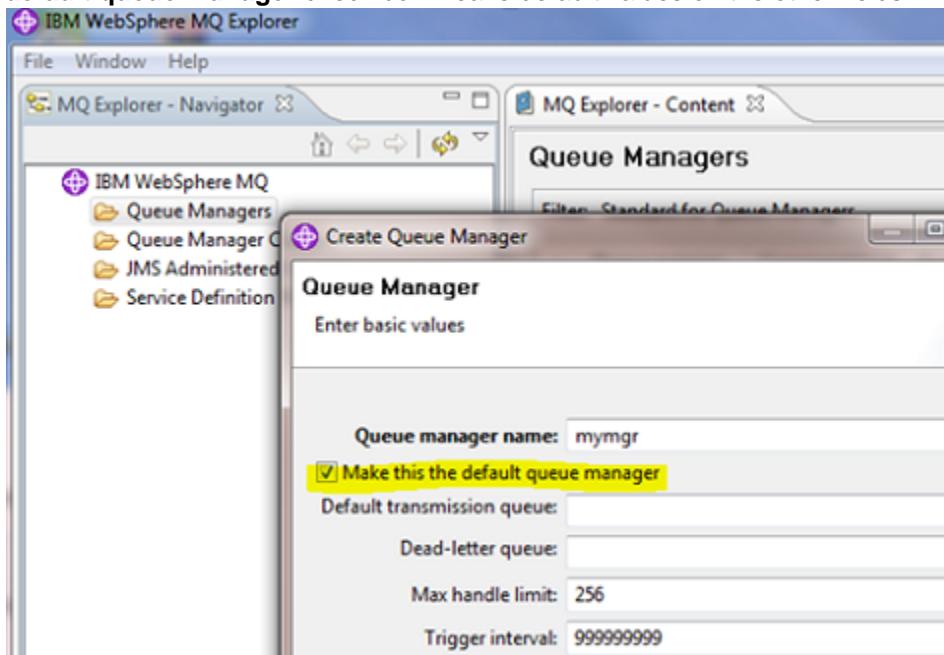
Your JMSAdmin.config file should now look similar to this:

```
# 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
.....
```

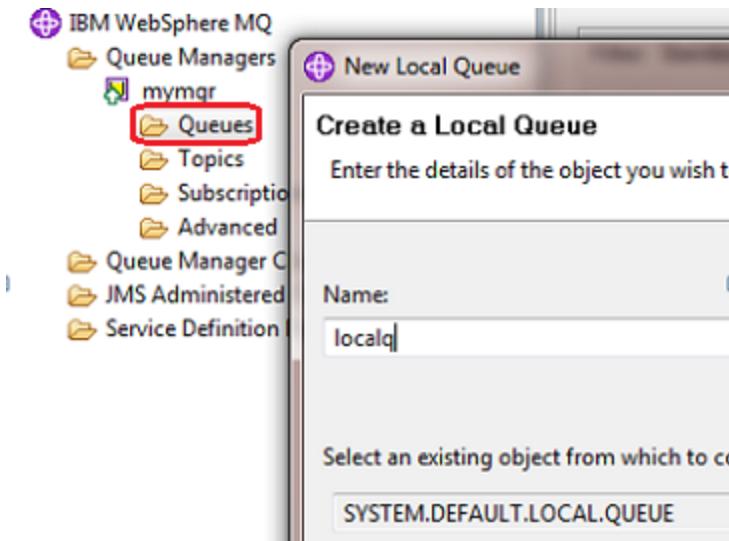
4. Restart the WebSphere MQ service.

Creating the Queue in WebSphere MQ

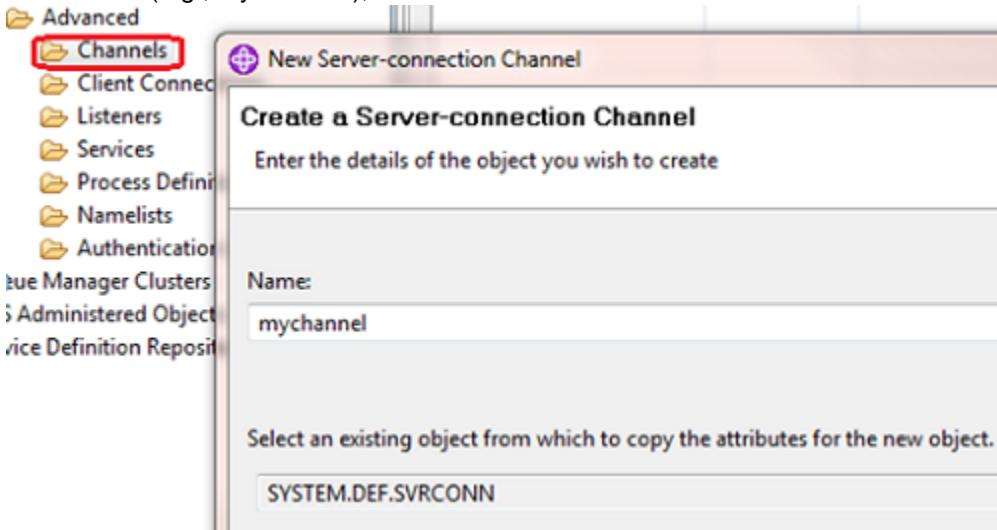
1. Start IBM WebSphere MQ Explorer and create a new queue manager. Make sure you select **make this the default queue manager** check box. Leave default values on the other fields.



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
+ 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.ConnectionFactoryJNDI</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
---------------	---	----------------------	---

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 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 <table border="1"> <tr> <td>JNDI Initial Context Factory Class*</td> <td>org.apache.qpid.jndi.PropertiesFileInitialContextFactory ② JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</td> </tr> <tr> <td>JNDI Provider URL*</td> <td>repository/conf/jndi.properties ② URL of the JNDI provider.</td> </tr> <tr> <td>Username</td> <td>jms-user</td> </tr> <tr> <td>Password</td> <td>*****</td> </tr> <tr> <td>Connection Factory JNDI Name*</td> <td>TopicConnectionFactory ② The JNDI name of the connection factory.</td> </tr> <tr> <td>Destination Type*</td> <td>topic ② Type of the destination.</td> </tr> <tr> <td>Destination*</td> <td>test_topic</td> </tr> <tr> <td colspan="2"> Dynamic Adapter Properties <table border="1"> <tr> <td>Header</td> <td>header_name1:header_value1,header_name2:header_value2 ② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)</td> </tr> </table> </td> </tr> </table>		JNDI Initial Context Factory Class*	org.apache.qpid.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 <table border="1"> <tr> <td>Header</td> <td>header_name1:header_value1,header_name2:header_value2 ② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)</td> </tr> </table>		Header	header_name1:header_value1,header_name2:header_value2 ② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)
JNDI Initial Context Factory Class*	org.apache.qpid.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 <table border="1"> <tr> <td>Header</td> <td>header_name1:header_value1,header_name2:header_value2 ② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)</td> </tr> </table>		Header	header_name1:header_value1,header_name2:header_value2 ② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)																
Header	header_name1:header_value1,header_name2:header_value2 ② Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)																		
Mapping Configuration <table border="1"> <tr> <td>Message Format*</td> <td>text ② Select the output message format</td> </tr> <tr> <td colspan="2"> Advanced </td> </tr> </table>		Message Format*	text ② Select the output message format	Advanced															
Message Format*	text ② Select the output message format																		
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="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">JP4yDiEh6HogOEjJzQQwHaJFIWZlnJTzaERl4eYrwukNeypm36R+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.ConnectionFactoryJNDI</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 in WSO2 CEP Documentation.

- [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 ⓘ 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 <table border="1"> <tr> <td>JNDI Initial Context Factory Class*</td> <td>org.wso2.andes.jndi.PropertiesFileInitialContextFactory ⓘ JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</td> </tr> <tr> <td>JNDI Provider URL*</td> <td>repository/conf/jndi.properties ⓘ URL of the JNDI provider.</td> </tr> <tr> <td>Username</td> <td>jms-user</td> </tr> <tr> <td>Password</td> <td>*****</td> </tr> <tr> <td>Connection Factory JNDI Name*</td> <td>TopicConnectionFactory ⓘ The JNDI name of the connection factory.</td> </tr> <tr> <td>Destination Type*</td> <td>topic ⓘ Type of the destination.</td> </tr> <tr> <td>Destination*</td> <td>test_topic</td> </tr> <tr> <td colspan="2"> Dynamic Adapter Properties <table border="1"> <tr> <td>Header</td> <td>header_name1:header_value1,header_name2:header_value2 ⓘ Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)</td> </tr> </table> </td> </tr> </table>		JNDI Initial Context Factory Class*	org.wso2.andes.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 <table border="1"> <tr> <td>Header</td> <td>header_name1:header_value1,header_name2:header_value2 ⓘ Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)</td> </tr> </table>		Header	header_name1:header_value1,header_name2:header_value2 ⓘ Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)
JNDI Initial Context Factory Class*	org.wso2.andes.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 <table border="1"> <tr> <td>Header</td> <td>header_name1:header_value1,header_name2:header_value2 ⓘ Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)</td> </tr> </table>		Header	header_name1:header_value1,header_name2:header_value2 ⓘ Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)																
Header	header_name1:header_value1,header_name2:header_value2 ⓘ Define Transport Headers (eg header_name1:header_value1,header_name2:header_value2)																		
Mapping Configuration <table border="1"> <tr> <td>Message Format*</td> <td>text ⓘ Select the output message format</td> </tr> <tr> <td colspan="2"> Advanced </td> </tr> </table>		Message Format*	text ⓘ Select the output message format	Advanced															
Message Format*	text ⓘ Select the output message format																		
Advanced																			
<input type="button" value="Add Event Publisher"/>																			

- i** 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="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">JP4yDiEh6HogOEjJzQQwHaJFIWZlnJTzaERl4eYrwukNeypm36R+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.ConnectionFactoryJNDI</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_vali
---------------	---	----------------------	---

Related samples

For more information on WSO2 Message Broker event publisher type, see the following sample in WSO2 CEP Documentation.

- 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

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

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:value1,property_name2:value2 ... })

Dynamic Adapter Properties

Topic*

Mapping Configuration

Message Format* ⓘ Select the output message format

+ Advanced

Add Event Publisher

i 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="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 in WSO2 CEP Documentation.

- [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*	LoggerOutputEventAdapter 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*	logger ⑦ Select the type of Adapter to publish events
Dynamic Adapter Properties	
Unique Identifier	log_id ⑦ To uniquely identify a log entry
Mapping Configuration	
Message Format*	text ⑦ Select the output message format
+ Advanced	
<input type="button" value="Add Event Publisher"/>	

① 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="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 in WSO2 CEP Documentation.

- 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

sensor id int

Stream Attributes

To

Output Event Adapter Type* mqtt Select the type of Adapter to publish events

Static Adapter Properties

Client Id	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*	tcp://localhost:1883 <small>MQTT broker url</small>
Username	mqtt-user <small>Username of the broker (if required)</small>
Password	mqtt-password <small>Password of the broker (if required)</small>
Clean Session	true <small>Persist topic subscriptions and ack positions across client sessions</small>
Quality of Service*	1

Dynamic Adapter Properties

Topic*	test_topic
--------	------------

Mapping Configuration

Message Format*	xml <small>Select the output message format</small>
-----------------	---

Advanced

Add Event Publisher

- i** 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="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 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 in WSO2 CEP Documentation.

- 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](#).

i 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.

i 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
Stream Attributes	
sensor id int	
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	

[Add Event Publisher](#)

i 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](#) .

- ✓** 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. i 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 in WSO2 CEP Documentation.

- 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; margin-top: -10px;">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: -10px;">⑦ The stream of events that need to be published</div> <div style="border: 1px solid #ccc; padding: 5px; height: 100px; vertical-align: top;">sensor id int</div>						
Stream Attributes <div style="border: 1px solid #ccc; height: 100px;"></div>							
To Output Event Adapter Type* <input type="text" value="sms"/> <div style="font-size: small; margin-top: -10px;">⑦ Select the type of Adapter to publish events</div>							
Dynamic Adapter Properties <table border="0" style="width: 100%;"> <tr> <td style="width: 15%;">Phone No*</td> <td><input type="text" value="0716453453"/></td> </tr> <tr> <td></td> <td style="font-size: small;">⑦ Phone No where SMS needs to be send (eg: [country-code][number])</td> </tr> </table>		Phone No*	<input type="text" value="0716453453"/>		⑦ Phone No where SMS needs to be send (eg: [country-code][number])		
Phone No*	<input type="text" value="0716453453"/>						
	⑦ Phone No where SMS needs to be send (eg: [country-code][number])						
Mapping Configuration <table border="0" style="width: 100%;"> <tr> <td style="width: 15%;">Message Format*</td> <td><input type="text" value="text"/></td> </tr> <tr> <td></td> <td style="font-size: small;">⑦ Select the output message format</td> </tr> <tr> <td style="text-align: right; padding-right: 10px;"> Advanced</td> <td></td> </tr> </table>		Message Format*	<input type="text" value="text"/>		⑦ Select the output message format	 Advanced	
Message Format*	<input type="text" value="text"/>						
	⑦ Select the output message format						
 Advanced							

[Add Event Publisher](#)

i 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="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.

i 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 in WSO2 CEP Documentation.

- [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> </tr> <tr> <td>User Name</td> <td><input type="text" value="soap-user"/></td> </tr> <tr> <td>Password</td> <td><input type="password" value="*****"/></td> </tr> <tr> <td>SOAP Headers</td> <td><input type="text" value="header1: value1, header2: value2"/> ⑦ Specify necessary SOAP headers. (e.g. "header1: value1, header2: value2")</td> </tr> <tr> <td>HTTP Headers</td> <td><input type="text" value="header1: value1, header2: value2"/> ⑦ Specify necessary HTTP headers. (e.g. "header1: value1, header2: value2")</td> </tr> </table> </div>		Url*	<input type="text" value="http://localhost:9763/services/Axis2LogService/log"/>	User Name	<input type="text" value="soap-user"/>	Password	<input type="password" value="*****"/>	SOAP Headers	<input type="text" value="header1: value1, header2: value2"/> ⑦ Specify necessary SOAP headers. (e.g. "header1: value1, header2: value2")	HTTP Headers	<input type="text" value="header1: value1, header2: value2"/> ⑦ Specify necessary HTTP headers. (e.g. "header1: value1, header2: value2")
Url*	<input type="text" value="http://localhost:9763/services/Axis2LogService/log"/>										
User Name	<input type="text" value="soap-user"/>										
Password	<input type="password" value="*****"/>										
SOAP Headers	<input type="text" value="header1: value1, header2: value2"/> ⑦ Specify necessary SOAP headers. (e.g. "header1: value1, header2: value2")										
HTTP Headers	<input type="text" value="header1: value1, header2: value2"/> ⑦ Specify necessary HTTP headers. (e.g. "header1: value1, header2: value2")										
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											

i 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="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 in CEP Documentation.

- [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 in CEP Documentation.

- 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.

- 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](#)

i 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 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 in CEP Documentation.

- [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.

- [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/input-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: -10px;">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: -10px;">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="websocket-local"/> <div style="font-size: small; margin-top: -10px;">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: -10px;">⑦ Select the output message format</div>	
+ Advanced	

[Add Event Publisher](#)

Related samples

For more information on websocket-local event publisher type, see the following sample in CEP Documentation.

- 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="font-size: small; 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> <pre>sensor id int</pre>
Stream Attributes	
To Output Event Adapter Type* <input type="text" value="wso2event"/> <div style="font-size: small; color: #ccc;">⑦ Select the type of Adapter to publish events</div>	
Static Adapter Properties	
Receiver URL* <input type="text" value="tcp://localhost:7661"/> <div style="font-size: small; color: #ccc;">⑦ Enter the Receiver Url</div>	
Authenticator URL <input type="text" value="tcp://auth-host:7661"/> <div style="font-size: small; color: #ccc;">⑦ Enter the Authenticator Url</div>	
User Name* <input type="text" value="wso2event-user"/> <div style="font-size: small; color: #ccc;">⑦ Enter the UserName</div>	
Password* <input type="password" value="*****"/> <div style="font-size: small; color: #ccc;">⑦ Enter the Password</div>	
Protocol* <input type="text" value="thrift"/> <div style="font-size: small; color: #ccc;">⑦ The communication protocol that will be used to published events</div>	
Publishing Mode <input type="text" value="non-blocking"/> <div style="font-size: small; color: #ccc;">⑦ Select the how events should be published</div>	
Publishing Timeout <input type="text" value="0"/> <div style="font-size: small; color: #ccc;">⑦ Timeout for the non-blocking Publishing Mode, default its '0' (fail immediately)</div>	
Mapping Configuration	
Message Format* <input type="text" value="wso2event"/> <div style="font-size: small; color: #ccc;">⑦ Select the output message format</div>	
+ Advanced	

Add Event Publisher

- i** 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 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 in WSO2 CEP Documentation.

- 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

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. A sample mapping configuration is shown below.

Mapping Configuration

Message Format*

wso2event

 Select the output message format

+
Advanced

WSO2Event Mapping

Output Event Stream *	<input type="text" value="sensor.stream"/>									
Output Event Version *	<input type="text" value="1.0.7"/>									
Meta Data <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <thead> <tr> <th>Name</th> <th>Value Of</th> <th>Actions</th> </tr> </thead> <tbody> <tr> <td>id</td> <td>meta_sensorId</td> <td> Delete</td> </tr> <tr> <td>name</td> <td>meta_sensorName</td> <td> Delete</td> </tr> </tbody> </table>		Name	Value Of	Actions	id	meta_sensorId	Delete	name	meta_sensorName	Delete
Name	Value Of	Actions								
id	meta_sensorId	Delete								
name	meta_sensorName	Delete								
Name :	<input type="text"/> Value Of : <input type="text" value="meta_sensorName"/> Add									

Correlation Data

No Correlation Data properties Defined

Name :	<input type="text"/>	Value Of :	<input type="text" value="meta_timestamp"/>	Add
--------	----------------------	------------	---	-----

Payload Data

Name	Value Of	Actions
humidity	humidity	Delete
value	sensorValue	Delete

Name :	<input type="text"/>	Value Of :	<input type="text" value="sensorValue"/>	Add
--------	----------------------	------------	--	-----

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

Copyright © WSO2 Inc. 2005-2014

365

```

<?xml version="1.0" encoding="UTF-8"?>
<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 - CEP will send this event as a WSO2Event with a null value for the particular attribute
- **String Attributes**
 - null - It will be sent out as a null value
 - Empty Attribute - It will be published as an empty string attribute

XML output mapping

XML output mapping converts canonical events of the server in the WSO2Event format to any XML message format that an endpoint can support. 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.

```
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
    <from ... />
    <mapping customMapping="enable" type="xml">
        <inline>
            <sensorData xmlns="">
                <id>{{meta_sensorId}}</id>
                <sensorValue>{{sensorValue}}</sensorValue>
                <humidity>{{humidity}}</humidity>
            </sensorData>
        </inline>
    </mapping>
    <to ... />
</eventPublisher>
```



Events with null or empty attributes

In case of an event is being published with null or empty attributes the behaviour for XML Output mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
 - null - CEP will convert this to an empty value and send out
 - Empty attribute - CEP will send this as it is as an empty value
- **String Attributes**
 - null - It will be converted to an empty string and sent out
 - Empty Attribute - It will be published as an empty string attribute

JSON output mapping

JSON output mapping converts canonical events of the server in the WSO2Event format to any JSON message format that an endpoint can support. A sample mapping configuration is shown below.

Mapping Configuration

Message Format* json Select the output message format

+ Advanced

JSON Mapping

OutputMapping Content* In-Line Pick from registry

```
{"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.

```
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
    <from ... />
    <mapping customMapping="enable" type="json">
        <inline>{"sensorData": {
            "id": {{meta_sensorId}},
            "value": {{sensorValue}},
            "humidity": {{humidity}},
            "correlation": {"long": {{correlation_longitude}}, "lat": {{correlation_latitude}}}
        }}</inline>
    </mapping>
    <to ... />
</eventPublisher>
```



Events with null or empty attributes

In case of an event is being published with null or empty attributes the behaviour for JSON Output mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
 - null - CEP will send this as a null value
 - Empty attribute - CEP will send this as it is as an empty value
- **String Attributes**
 - null - It will be sent out as a null value
 - 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. A sample mapping configuration is shown below.

Mapping Configuration

Message Format*   Select the output message format

 Advanced

Text Mapping

OutputMapping Content* In-Line Pick from registry

```
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.

```
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
    <from ... />
    <mapping customMapping="enable" type="text">
        <inline>Sensor Data
        Sensor ID : {{meta_sensorId}}
        Sensor Name : {{meta_sensorName}}
        Sensor located at ({{correlation_longitude}}, {{correlation_latitude}})
        Value : {{sensorValue}}
        Humidity : {{humidity}}
    </inline>
    </mapping>
    <to ... />
</eventPublisher>
```



Events with null or empty attributes

In case of an event is being published with null or empty attributes the behaviour for Text Output mapping is as follows.

- **Non-String Attributes** (e.g. int, double, long etc.)
 - null - CEP will convert this to an empty value and send out
 - Empty attribute - CEP will send this as it is as an empty value
- **String Attributes**
 - null - It will be converted to an empty string and sent out
 - Empty Attribute - It will be published as an empty string attribute

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 ConfigurationMessage Format*  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 : 

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

```
<?xml version="1.0" encoding="UTF-8"?>
<eventPublisher ... xmlns="http://wso2.org/carbon/eventpublisher">
    <from ... />
    <mapping customMapping="enable" type="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 - CEP will send this as a null value
 - Empty attribute - CEP will send this as it is as an empty value
- **String Attributes**
 - null - It will be sent out as a null value
 - Empty Attribute - It will be published as an empty string attribute

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.

 Import jquery.js first to use carbon-analytics.js.

```
var client = new AnalyticsClient().init();

default username: admin
default password: admin
server_url : https://localhost:9443/portal/controllers/apis/analytics.jag
```

Above constructor creates an object of the AnalyticsClient module with default parameters. If you want to create an analytics client with your own parameters, use the below constructor.

```
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["message"]);
});
```

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[ "message" ]);
});
```

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["message"]);
});
```

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["message"]);
});
```

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["message"]);
});
```

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["message"]);
});

```

Sample output

```
[ {
    "id": "14399961246350.13125980939205978",
    "tableName": "sampleTable", "timestamp": 1439996124610,
    "values": {
        "column1": "234,454,34\u0027",
        "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 occured: " + error["message"]);
});

```

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["message"]);
});
```

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["message"]);
});
```

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 : "TEST", //table being queried
    query : <lucene-query> //lucene query to search the records
};
client.searchCount(queryInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occured: " + error["message"]);
});
```

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.
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
    query : "column1:value1", //lucene query to search the records
    start : 0, //starting index of the matching record set
    count : 100 //page size for pagination
};
client.search(queryInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error["message"]);
});
```

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 occured: " + error["message"]);
});
```

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 occured: " + error["message"]);
});
```

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["message"]);
});
```

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["message"]);
});

```

⚠ 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
        scoreFunction : "scoreParamField * 2" //score function, optional
    };
}

client.drillDownSearch(drillDownReq, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error["message"]);
});

```

Sample output

```
[
{
    "id": "14242656601230.5739142271249453",
    "tableName": "TEST",
    "timestamp": 1424265660123,
    "values": {
        "timestamp": "value1",
        "product": "wso2cdm",
        "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(drillDown, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occurred: " + error["message"]);
});

```

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["message"]);
});

```

Sample output

```
TEST:1.0.0
```

Publishing Events to WSO2 DAS via JS API

- Overview
- Example
- Sample output

Overview

Function	publish(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 = {
    streamId : "TEST:1.0.0",
    timestamp : 54326543254532, "optional"
    payloadData : {
    },
    metaData : {
    },
    correlationData : {
    },
    arbitraryDataMap : {
    }
};
client.publish(event, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occured: " + error["message"]);
});

```

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.publish(requestInfo, function(data) {
    console.log (data["message"]);
}, function(error) {
    console.log("error occured: " + error["message"]);
});

```

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["message"]);
});
```

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 = {
    searchParams : {
        tableName:"TEST",
        groupByField:"single_valued_facet_field",
        aggregateFields:[
            {
                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["message"]);
});
```

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	eventstreams
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

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 4 different kind of artifact event streams, event receiver, event store, Spark scripts.

```

artifacts.xml
Eventreceiver_1.0.0
    artifact.xml
    TestWso2EventReceiver.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
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.

```
<?xml version="1.0" encoding="UTF-8"?><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="Eventstream" version="1.0.0" include="true"
serverRole="DataAnalyticsServer"/>
    <dependency artifact="Sparkscripts" 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"/>
            </dependency>
        </dependency>
    </dependency>
</artifacts>
```

2. Create separate directories in the top level of the C-App for the above mentioned four dependencies (`Eventreceiver_1.0.0`, `Eventstore_1.0.0`, `Eventstream_1.0.0`, and `Sparkscripts_1.0.0`) defined in the `artifacts.xml` file.

i 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

```
<?xml version="1.0" encoding="UTF-8"?>
<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

```
<?xml version="1.0" encoding="UTF-8"?><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

```
<?xml version="1.0" encoding="UTF-8"?><artifact name="Eventreceiver" version="1.0.0" type="event/receiver" serverRole="DataAnalyticsServer">
<file>TextWso2EventReceiver.xml</file>
</artifact>
```

Sparkscripts_1.0.0 artifact.xml

```
<?xml version="1.0" encoding="UTF-8"?><artifact name="Sparkscripts" version="1.0.0" type="analytics/spark" serverRole="DataAnalyticsServer">
<file>sample_script.xml</file>
</artifact>
```

Dashboard_1.0.0 artifact.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<artifact name="Dashboard" version="1.0.0" type="dashboards/dashboard" serverRole="DataAnalyticsServer">
<file>appman.json</file>
</artifact>
```

Gadget_1.0.0 artifact.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<artifact name="Gadget" version="1.0.0" type="dashboards/gadget" serverRole="DataAnalyticsServer">
<file>Test_Gadget</file>
</artifact>
```

Layout_1.0.0 artifact.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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.test_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

```

<?xml version="1.0" encoding="UTF-8"?>
<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>

```

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>

```

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  "
            }
        }
    ],
    "content": {
        "d": [
            {
                "id": "24w6ukj9olz69a4i",
                "content": {
                    "id": "Temperatur_e_By_City",
                    "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"
                    },
                    "style": {
                        "title": "Temperature By City",
                        "borders": true
                    },
                    "options": {
                        "dataSource": {
                            "type": "STRING",
                            "title": "DataSource",
                            "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"}}
```

Laoyut - 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.

The screenshot shows the WSO2 Management Console interface for adding a Carbon Application. At the top, the URL is [Home > Manage > Carbon Applications > Add](#). Below it, the heading is **Add Carbon Applications**.

In the center, there is a form titled "Upload Carbon Application" with a "Browse..." button and a message "No file selected.". Below the form are two buttons: "Upload" and "Cancel".

A modal dialog titled "File Upload" is displayed over the form. It shows a file list with the following entries:

Name	Size	Modified
sparkscripts_1.0.0		16:21
Eventreceiver_1.0.0		14:32
Eventstream_1.0.0		14:29
Eventstore_1.0.0		14:25
DASTestCApp.car	3.7 kB	16:21
artifacts.xml	593 bytes	16:21

The file "DASTestCApp.car" is highlighted in orange. At the bottom of the dialog are "Cancel" and "Open" buttons.

At the very bottom of the page, the URL is again listed as [Home > Manage > Carbon Applications > Add](#), followed by the heading **Add Carbon Applications**.

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, there is a 'Help' link. Below the header, it says '1 Running Carbon Applications.' A table is displayed with three columns: 'Carbon Applications', 'Version', and 'Actions'. The first row contains 'DASTestCApp' in the 'Carbon Applications' column, '1.0.0' in the 'Version' column, and two buttons in the 'Actions' column: a blue 'Delete' button and a grey 'Download' button.

Carbon Applications	Version	Actions
DASTestCApp	1.0.0	Delete Download

5. Click on the **Delete** option to delete the Carbon application as shown below.

The screenshot shows the same 'Carbon Applications List' page as the previous one. The 'Delete' button in the 'Actions' column of the first row is highlighted with a red box. The rest of the interface is identical to the previous screenshot.

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.

i 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.

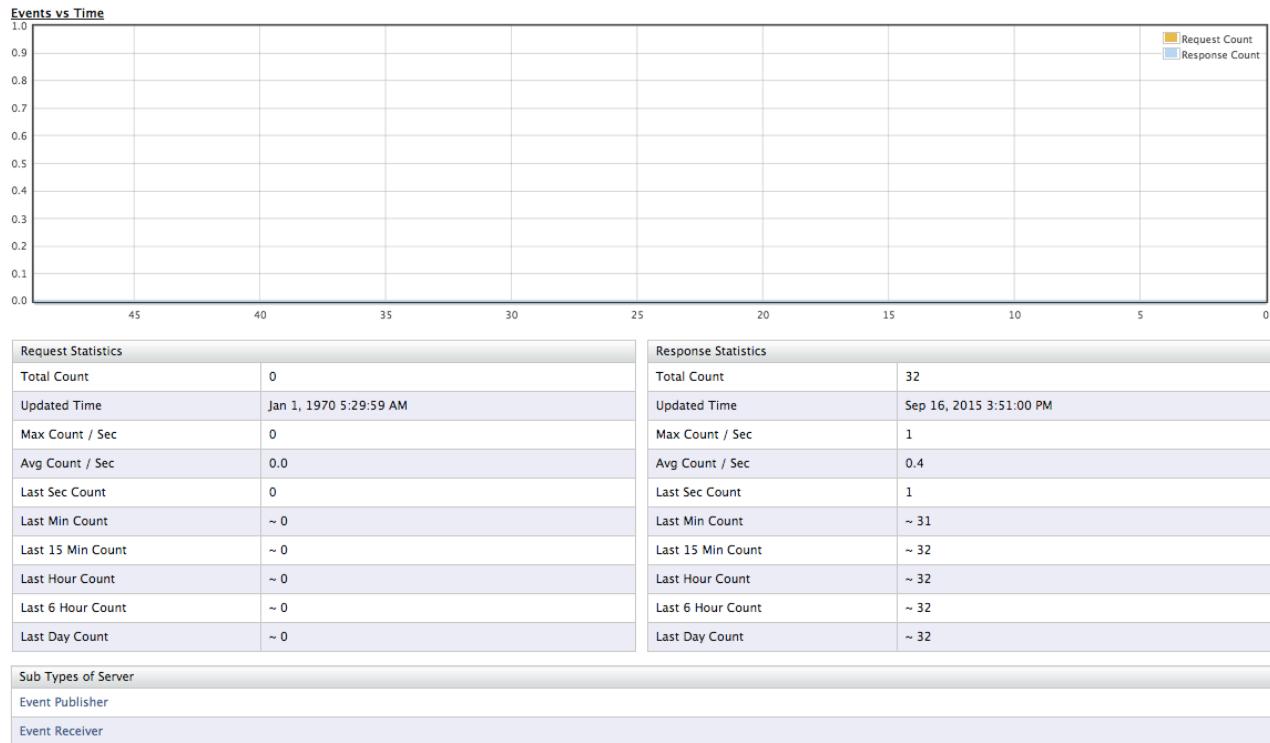
The screenshot shows a table with one row of data. The columns are: Event Publisher Name, Message Format, Output Event Adapter Type, Input Stream ID, and Actions. The data row is: logger, json, logger, org.wso2.event.sensor.stream:1.0.0, and Actions. The 'Actions' column contains several icons: a green plus sign for 'Add Event Publisher', a blue gear for 'Edit', a yellow trash can for 'Delete', and a red-bordered blue gear for 'Enable Statistics'. The 'Enable Statistics' icon is highlighted with a red box.

Event Publisher Name	Message Format	Output Event Adapter Type	Input Stream ID	Actions
logger	json	logger	org.wso2.event.sensor.stream:1.0.0	+ Add Event Publisher Edit Delete Enable Statistics

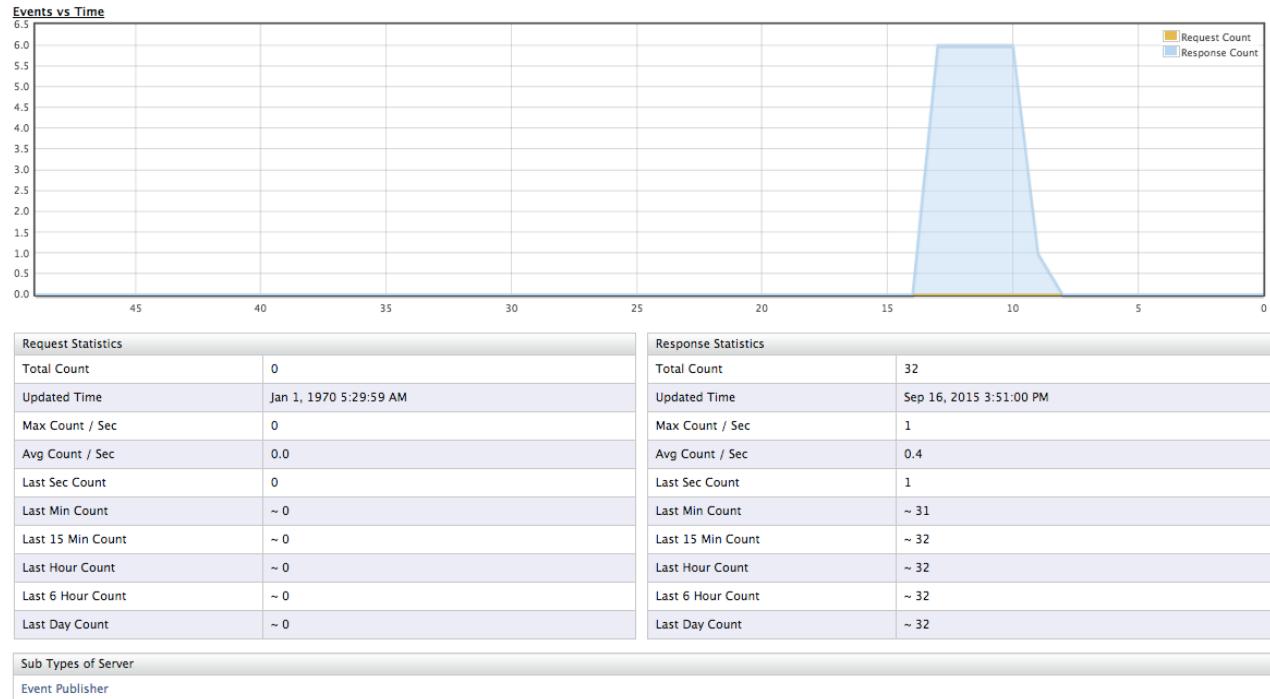
Monitoring Event Statistics

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

Available Input Event Adapters		
Add Input Event Adaptor 2 Active Input Adaptors. 0 Inactive Input Adaptors		
Event Adaptor Name	Event Adaptor Type	Actions
emailAdaptor	email	Enable Statistics Enable Tracing Delete Edit
WSO2EventAdaptor	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.

Event Message Tracer	
<input type="text"/> Search <input type="checkbox"/> Ignore Case	
... No trace entries found. You can enable tracing on transport adapters, event formatters, event builders or event processors by visiting the relevant pages ...	
Clear All	

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

```
<quotedata:CompanyName>Microsoft Corpora</quotedata:CompanyName>
<quotedata:QuoteError>false</quotedata:QuoteError>
</quotedata:StockQuoteEvent>
</quotedata:AllStockQuoteStream>
15:02:42,344 [-] [http-nio-9763-exec-1] INFO [Event-Builder] Received event as OMElement.
<quotedata:AllStockQuoteEvent xmlns:quotedata="http://ws.cdyne.com/">
  <quotedata:StockSymbol>MSFT</quotedata:StockSymbol>
  <quotedata>LastTradeAmount>26.36</quotedata>LastTradeAmount>
  <quotedata:StockChange>0.05</quotedata:StockChange>
  <quotedata:OpenAmount>25.05</quotedata:OpenAmount>
  <quotedata:DayHigh>25.46</quotedata:DayHigh>
  <quotedata:DayLow>25.01</quotedata:DayLow>
  <quotedata:StockVolume>20452658</quotedata:StockVolume>
  <quotedata:PrevCls>25.31</quotedata:PrevCls>
  <quotedata:ChangePercent>-0.20</quotedata:ChangePercent>
  <quotedata:FiftyTwoWeekRange>22.73 - 31.58</quotedata:FiftyTwoWeekRange>
  <quotedata:EarnPerShare>2.32</quotedata:EarnPerShare>
  <quotedata:PE>10.88</quotedata:PE>
  <quotedata:CompanyName>Microsoft Corpora</quotedata:CompanyName>
  <quotedata:QuoteError>false</quotedata:QuoteError>
</quotedata:StockQuoteEvent>
</quotedata: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:1.0.0
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

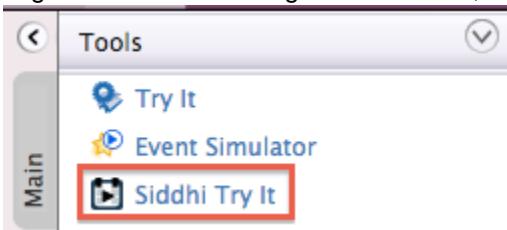
```
15:02:42,346 [-] [http-nio-9763-exec-1] INFO Events arrived at junction. Event:[Event{streamId='newStockQuoteStream', timeStamp=1375349562345, data=[26.36, MSFT], type=new}] Stream:stockStream:1.0.0
15:02:42,346 [-] [http-nio-9763-exec-1] INFO Dispatching events to the event formatter. Events: [Event{streamId='newStockQuoteStream', timeStamp=1375349562345, data=[26.36, MSFT], type=new}]
15:02:42,370 [-] [http-nio-9763-exec-3] 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	The event processing logic written in Siddhi QL. When defining more than one query, end each query with a semicolon. Defining a query name (e.g. query1) is optional.	@info(name = 'query1') f r o m sensorStream[temperature>98.6] select sensorId insert into outputStream;
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

- Enter a time stamp to begin the process of sending events for **Begin Time** if required.
- 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]
```

5. Click **Submit**.

You view the input stream and the results of the execution plan under the defined output stream, and separated query outputs as shown below.

Result

```
[ {
    "timestamp": 1432039938174,
    "data": [
        "tempID1"
    ],
    "isExpired": false
}]
```

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](#)
- [Installing Machine Learner Features for the CEP Extension](#)

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](#)
- [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.common
- org.wso2.carbon.analytics.dataservice.core
- org.wso2.carbon.analytics.datasource.cassandra
- org.wso2.carbon.analytics.datasource.common
- 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.common
- 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.common

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/externa-l-spark-classpath.conf file in a UNIX environment.

-  Each path should have a separate line.

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> • 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> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  This property is applicable only when the Spark master runs in the cluster mode. </div>
carbon.das.symbolic.link	This links to your DAS home by default.	<p>The symbolic link for the jar files in the Spark class path.</p> <p>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>.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  The symbolic link is not specified by default. When it is not specified, the jar files are added in the DAS home. </div>

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

i 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

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.
sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableEventSink=true	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.	You can use this property when you want to have a node only as a real time analytics node.
sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableDataPurging=true	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 message console.	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.

sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableAnalyticsSparkCtx=true	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.	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.
sh <PRODUCT_HOME>/bin/wso2server.sh -DenableAnalyticsStats=true	If this system property is set, then the Spark query execution statistics are printed in the Carbon Console.	This property can be used when you need to view the Analytics statistics in the Carbon Console.
s h <PRODUCT_HOME>/bin/wso2server.sh -DenableIndexingStats=true	If this system property is set, statistics of the background indexing tasks are printed in the Carbon console.	
sh <PRODUCT_HOME>/bin/wso2server.sh-D disableIndexThrottling=true		

i 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 **Message Console**.
3. Select a **Table Name**, and click **Schedule Data Purging** as shown below.



The **Schedule Data Purging** option is displayed only for users of whom the assigned role has the **De**

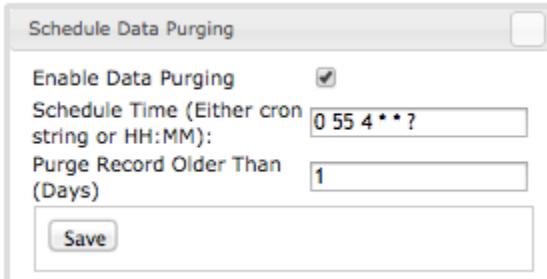
lete permission under Record enabled. For instructions on setting this permission to users, see [WSO2 DAS-Specific User Permissions](#).

Home > Manage > Interactive Analytics > Message Console

Message Console

The screenshot shows a search interface with a 'Table Name*' dropdown set to 'JMX_AGENT_TOOLBOX'. Below it are two radio buttons: 'By Date Range' and 'By Query'. At the bottom are 'Search' and 'Reset' buttons. A red box highlights the 'Schedule Data Purging' button.

- Enter the details to schedule the data purging task as shown below in the screen which pops up.



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 pop up screen.

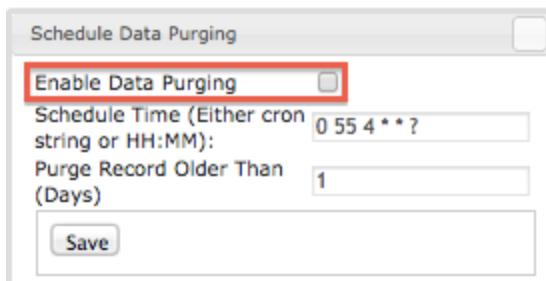


If you want to remove a data purging operation which you scheduled, click

Removing scheduled data purging operations

Follow the steps below to remove a data purging operation which you already scheduled on a selected table.

- Log in to the management console as an admin user, if you are not already logged in.
- Click **Main**, and then click **Message Console**.
- Select a **Table Name**, and click **Schedule Data Purging** as shown below.
- Deselect **Enable Data Purging** as shown below.



5. Click **Save**, and close the pop up screen.

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.

Home > Registry > Browse

Browse

Root /

Location: /

Tree view Detail view

The screenshot shows the WSO2 Data Analytics Server Registry interface. The left pane displays a hierarchical tree view of registry entries under the root. The tree includes categories like system, config, governance, event, permission, repository, and StreamDefinitions. Under StreamDefinitions, there are sub-folders for bam_mediation_stats_data_publisher, BAM_MESSAGE_TRACE, BAM_MESSAGE_TRACE_FILTER, HttpEventStream (which is expanded), WSO2EventEventStream, StreamIndexDefinitions, trunk, and local. The '1.0.0' version of the HttpEventStream is specifically highlighted with a red rectangular box around its icon and name.

/

_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.

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 [Carbon Application Deployment for DAS](#). 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 backup 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	Examples
----------	---------	----------

-backupRecordStore	To backup the record store where the persisted events are saved. This	<pre>./analytics-backup.sh -backupRecordStore - d i r /home/user/backup</pre> The above command backs up all the tables in the DAS record store to the /home/user/backup directory.
-backupFileSystem	To backup the file system where data relating to the indexing of events.	<pre>./analytics-backup.sh -backupFileSystem - d i r /home/user/backup</pre> The above command backs up indexing information relating to all the tables in DAS to the /home/user/backup directory.
-dir <directory>	Directory used as the target when backing up the record store/file system, or as the source when restoring already backed up record store/file system. The directly should be specified in every command issued to backup or restore data.	<pre>- d i r /home/user/backup</pre> The above argument in all the other examples specifies the /home/user/backup directory as the location where backed up data should be stored or as the source from which data should be restored.
-restoreRecordStore	To restore a record store. Before you use this command, the record store you want to restore should be already backed up.	<pre>./analytics-backup.sh -restoreRecordStore - d i r /home/user/backup</pre> The above command restores record store data currently saved in the /home/user/backup directory to the current DAS node.
-restoreFileSystem	To restore a file system. Before you use this command, the file system you want to restore should be already backed up.	<pre>./analytics-backup.sh -restoreFileSystem - d i r /home/user/backup</pre> The above command restores indexing related data currently saved in the /home/user/backup directory to the current DAS node.

-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 corruption. If you are certain that the target DAS server is not running, you can index the data at the time it is restored by using this argument. If this argument is not used while the target DAS server is not running, the data restored from the time the target server is started up is indexed in the usual way.</p>	<pre>./analytics-backup.sh -restoreRecordStore -restoreFileSystem -enableIndexing -dir /home/user/backup</pre> <p>The above command restores both record store data and indexing related data currently saved in the /home/user/backup directory to the current DAS node, and indexes the data at the time of restoration.</p>
-reindexEvents	<p>This switch will make the tool re-index the data already there in a table. This can be useful in a scenario where an the indexing is corrupted for some reason, or if some of the older data is not indexed because it was not mentioned in the schema etc..</p>	<pre>./analytics-backup.sh -restoreRecordStore -restoreFileSystem -reindexEvents -dir /home/user/backup</pre> <p>The above command restores both record store data and indexing related data currently saved in the /home/user/backup directory to the current DAS node, and reindexes the event at the time of restoration.</p>
-tables <table list>	<p>This argument is used to specify the list of event tables that should be backed up or restored.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> If you do not use this argument to back up data from one or more specific tables, all the available event tables will be backed up. </div>	<pre>/analytics-backup.sh -backupRecordStore -tables Table1,Table2,Table3 -dir /home/user/backup</pre> <p>The above command backs up the data of Table1, Table2, Table3 event tables to the /home/user/backup directory.</p> <pre>./analytics-backup.sh -restoreRecordStore -tables Table1,Table2,Table3 -dir /home/user/backup</pre> <p>The above command restores record store data of Table1, Table2, Table3 event tables that are currently saved in the /home/user/backup directory to the current DAS node.</p>

-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.	<pre>./analytics-backup.sh -backupRecordStore -tenantId abc -dir /home/user/backup</pre> The above command backs up all the event tables of the abc tenant from the DAS record store.
-disableStaging	This argument is used to disable staging. Staging involves including the indication whether the data is indexed or not when backing up/restoring the DAS record store. This task incurs a greater system overhead. Therefore, it is allowed to disable staging when the number of records being backed up/restored is high. Backing up and restoring the file system via -backupFileSystem and -restoreFileSystem allows you to preserve the indexing which would enable searches to be carried out on restored data.	<pre>./analytics-backup.sh -backupRecordStore -disableStaging</pre> The above command backs up all the event tables in the DAS record store. Indications of whether data is indexed or not are removed before backing up the records.
-timefrom <yy-mm-dd hh:mm:ss>	This argument specifies the starting time (inclusive) when defining the time period that should be considered when backing up/restoring events. <div style="border: 1px solid #ccc; padding: 5px; width: fit-content;">i This argument should be used together with the -timeto argument.</div>	<pre>./analytics-backup.sh -backupRecordStore -timefrom 15-10-11-12-40-00 -timeto 15-10-15-17-00-00 -dir /home/user/backup</pre> The above command backs up all the data in the DAS record store created between 12.45PM on 11th October 2015 to 5.00PM on 15th October 2015. The data is backed up in the /home/user/backup directory.
-timeto <yy-mm-dd hh:mm:ss>	This argument specifies the ending time (inclusive) when defining the time period that should be considered when backing up/restoring events. <div style="border: 1px solid #ccc; padding: 5px; width: fit-content;">i This argument should be used together with the -timefrom argument.</div>	<pre>./analytics-backup.sh -backupRecordStore -timefrom 15-10-11-12-40-00 -timeto 15-10-15-17-00-00 -dir /home/user/backup</pre> The above command backs up all the data in the DAS record store created between 12.45PM on 11th October 2015 to 5.00PM on 15th October 2015. The data is backed up in the /home/user/backup directory.

-  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 -restoreRecordStore -tables Table1,Table2,Table3
-backupFileSystem -timefrom 15-08-11-08-00-00 -timeto 15-10-15-22-30-00 -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 -tables Table1,Table2,Table3
-restoreRecordStore -timefrom 15-08-11-09-00-00 -timeto 15-08-11-10-00-00 -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.

Searching restored data

In order to enable searching for persisted data, WSO2 indexes events using Apache Lucene (for more information, see [Configuring Indexes](#)). The persisted data is saved in the record store and the information relating to the indices of that data is stored in the file system. Therefore, if you want to carry out searches once you restore backed up data, you need to restore both the record store and the file system. If you restore only the record store it will not be possible to carry out searches for the restored data using the Data Explorer. For more information on searching using the Data Explorer, see [Searching Data by Categories](#).

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](#)
- [WSO2 Carbon platform-level settings](#)
- [JDBC Pool Configuration](#)
- [DAS-Level settings](#)



Important

- Performance tuning requires you to modify important system files, which affect all programs running on the server. We recommend you to familiarize yourself with these files using Unix/Linux documentation before editing them.
- The parameter values we discuss below are just examples. They might not be the optimal values for

the specific hardware configurations in your environment. We recommend that you carry out load tests on your environment to tune the product accordingly.

OS-Level Settings

1. To optimize network and OS performance, configure the following settings in `/etc/sysctl.conf` file of Linux. These settings specify a larger port range, a more effective TCP connection timeout value, and a number of other important parameters at the OS-level.

```
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
```

i 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
```

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.
<code>testOnBorrow</code>	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 <code>validationQuery</code> property should be used if <code>testOnBorrow</code> is set to true. To increase the efficiency of connection validation and to improve performance, <code>validationInterval</code> property should also be used.
<code>validationInterval</code>	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 <code>testOnBorrow</code> 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.

- Receiving events
- Publishing events

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.
eventBufferCapacity	The size of the event receiving buffer. This buffer temporarily stores the events received before they are forwarded to an event stream .	10000	This value should be increased if you want to increase the throughput by receiving a higher number of events at a given time.

Property	Description	Recommendation
workerThreads	Number of threads in consumer to handle the load. Default value is 10.	Value should be higher when receiving throughput is high. Should to consider number of CPU cores.
eventBufferCapacity	Size of the receiving event buffer. Default value is 10000.	Needs to be higher value when receiving throughput is high. When increasing the value heap memory size also needs to be increased accordingly.

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>
```

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.

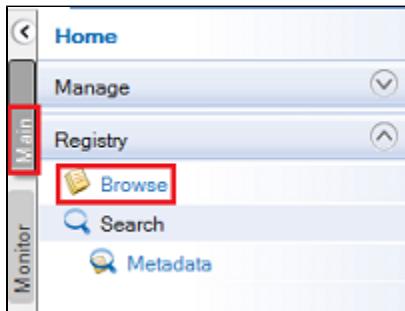
- **Governance Registry** - Used to store user-specified metadata and resources and can be shared across an organization.

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.

Browse

Root /

Location: /

Tree view Detail view

/ / _system

2. Click on a particular directory name to see its details in the *Detail view* tab.

Root / _system

Location: /_system

Tree view Detail view

Metadata

Properties

Entries

Add Resource
 Add Collection
 Create Link

Name	Created On	Author
config	06 Oct	wso2.system..
governance	06 Oct	wso2.system..
local	06 Oct	wso2.system..

Permissions



Note

You can access the root of the directory path by clicking the *Root* icon in the breadcrumb.

Browse

Root /

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,

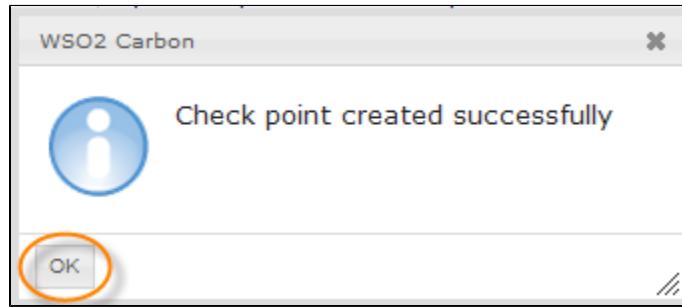
Metadata	
 Feed	
Created:	By wso2.system.user 13 May 12:05:58
Last Updated:	By wso2.system.user 27 May 12:05:39
Media Type:	Unknown
Checkpoint:	Create Checkpoint
Versions:	View versions
Description:	Governance registry of the carbon server. This collection is used to store the resources common to the 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:	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.
<input type="button" value="Save"/> <input type="button" value="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*.

Properties

Add New Property

Name*	prop2
Value*	value1

Add Cancel

1 Property

Name	Value	Action
prop1	val	Edit Delete

Editing a Property

1. Click on the *Edit* link of a particular property in the *Action* column.

Properties

Add New Property

2 Properties

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*.

Name	Value	Action
prop1	val	Save Cancel

Deleting a Property

1. To delete a property, click on the *Delete* link of a certain property in the *Action* column.

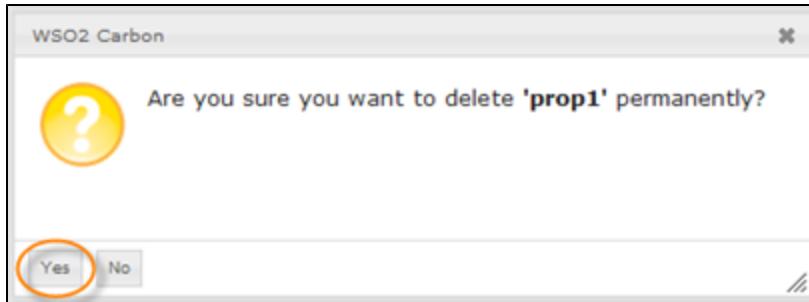
Properties

Add New Property

2 Properties

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..

Media Type: Unknown

Feed: Feed

Rating: ★★★★★

To see the available actions over a resource, click on the *Actions* icon.

Name	Created On	Author
config	28 Sep	wso2.system..

Actions

Operations: Rename, Move, Delete, Copy

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.

Name
CalculatorService.wsdl
Rename Move Delete Copy Download
Rename resource
New Resource Name * CalculatorService.wsdl
Rename Cancel

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*.

Name
CalculatorService.wsdl
Rename Move Delete Copy Download
Move resource
Destination Path *
Move Cancel

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.

Name
CalculatorService.wsdl
Rename Move Delete Copy Download

Copying a Resource

1. To copy a resource to some directory, click *Copy* and specify *Destination Path*.



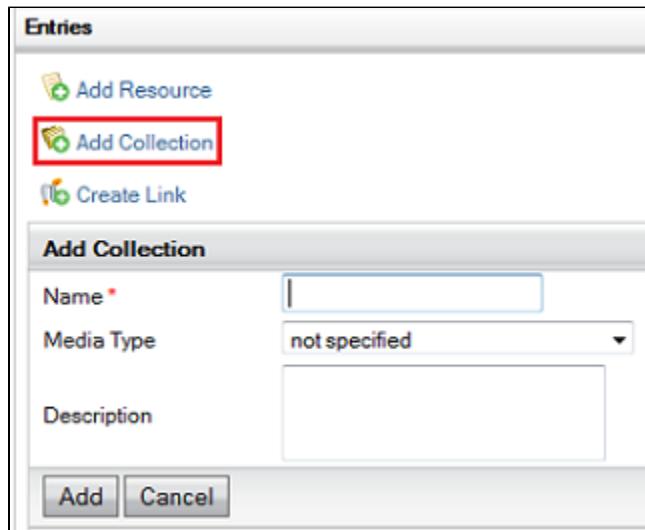
2. Click *Copy*.

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.

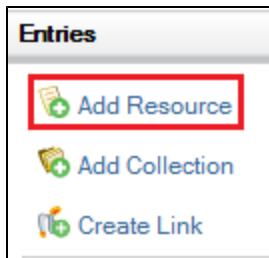
3. Click *Add*.

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 screenshot shows a dropdown menu for 'Method' with four options: 'Upload content from file', 'Import content from URL', 'Create Text content', and 'Create custom content'. The first option, 'Upload content from file', is highlighted with a red box. Below the dropdown, the text 'Uploading Content from File' is visible.

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 screenshot shows the 'Add Resource' dialog box for the 'Upload content from file' method. It includes fields for 'File' (with a browse button and placeholder 'C:\wso2\Chad.wsdl'), 'Name' ('Chad.wsdl'), 'Media type' ('application/wsdl+xml'), and 'Description' ('test'). The entire form area is highlighted with a red box. At the bottom are 'Add' and 'Cancel' buttons.

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:8280/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, it says 'Add Resource' and has a 'Method' dropdown set to 'Create custom content'. Below this 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'. At the bottom of the dialog are two buttons: '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 the panel is a 'Create Link' dialog box. It has a 'Method' dropdown set to 'Add Symbolic Link'. Underneath is a 'Add Symbolic Link' form with 'Name*' and 'Path*' fields. At the bottom of the dialog 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.

The screenshot shows the 'Create Link' dialog with the method set to 'Add Symbolic Link'. In the 'Add Symbolic Link' section, the 'Name' field is filled with 'symbol1' and the 'Path' field is filled with '/_system'. At the bottom are 'Add' and 'Cancel' buttons.

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.

The screenshot shows the 'Create Link' dialog with the method set to 'Add Remote Link'. In the 'Add Remote Link' section, the 'Name' field is filled with 'remote1' and the 'Remote Instance' dropdown shows 'No instances available'. The 'Path' field is empty. At the bottom are 'Add' and 'Cancel' buttons.

Managing Role Permissions

The **Permissions** panel shows the defined role permissions, allows to [add](#) new role permissions and [edit](#) existing ones.

The screenshot shows the 'Permissions' panel. The 'New Role Permissions' pane includes fields for 'Role' (set to 'wso2.anonymous.role'), 'Action' (set to '-Select-'), and 'Allow' (radio button selected). Below it is an 'Add Permission' button. The 'Defined Role Permissions' pane is a table:

Role	Read	Write	Delete	Authorize
	Allow	Deny	Allow	Deny
everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

At the bottom of the 'Defined Role Permissions' pane are 'Apply All Permissions' and 'Reset' buttons.

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.

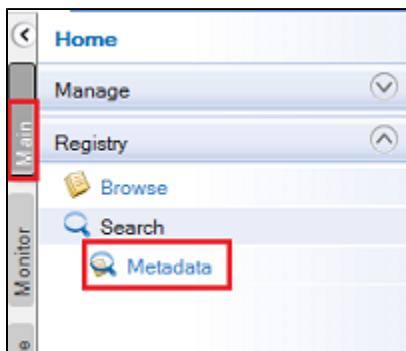


Tip

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"/>

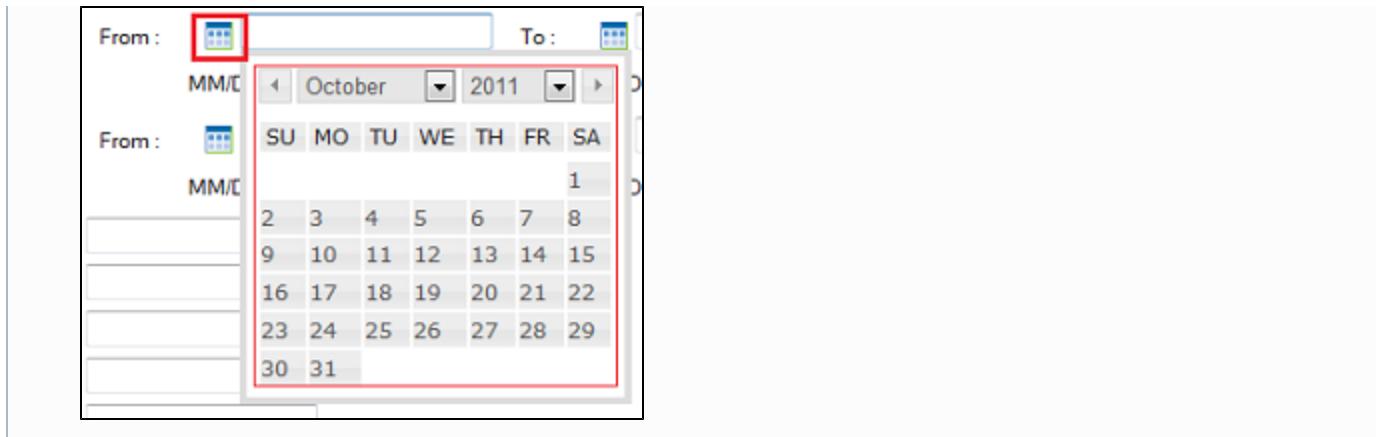
The search can be refined by:

- **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

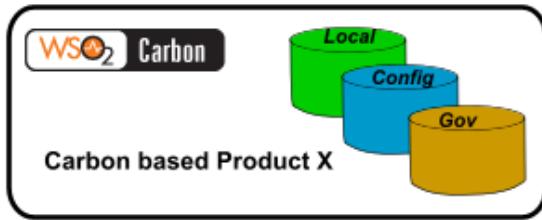
Pattern #1 : Local Registry

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.

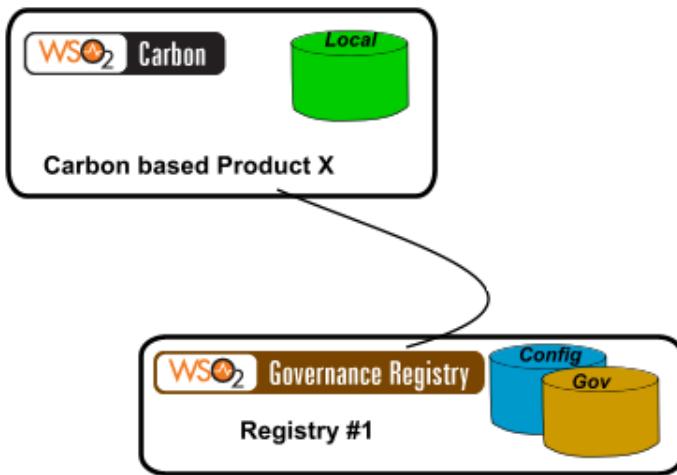
Pattern #2 : Remote Conf, Go.Reg

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 VM          : Java HotSpot(TM) 64-Bit Server VM EBS3500,Oracle Co
[2012-12-14 14:02:31,309] INFO - CarbonCoreActivator Carbon Home     : /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
[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,

The screenshot shows the 'Browse' interface of the WSO2 Data Analytics Server. At the top, there is a 'Root' node with a folder icon. Below it is a 'Location' input field containing a '/' character and a 'Go' button. Underneath these are two tabs: 'Tree view' (which is selected) and 'Detail view'. The main pane displays a hierarchical tree structure. The root node has several children: '_system', 'config' (which is highlighted with a red box), 'governance' (also highlighted with a red box), and 'local'. Each node is represented by a folder icon with a small blue arrow pointing to its right.

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.

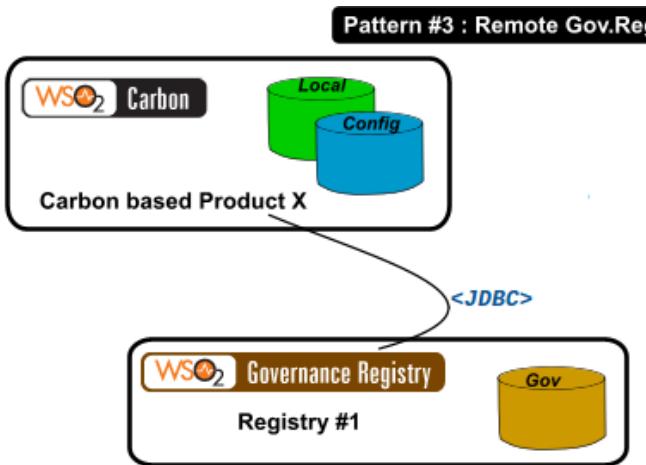


Figure 3: Governance partition in the remote Governance Registry instance.

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.

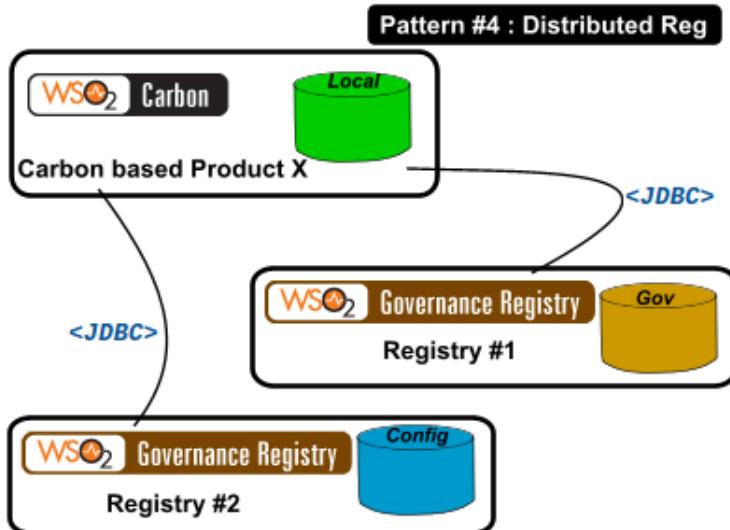


Figure 4: Config and governance partitions in separate registry instances.

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. Its 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.

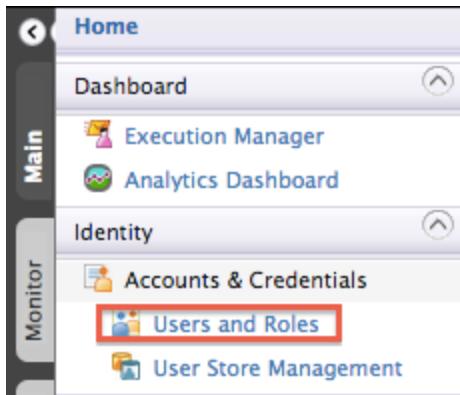
Go to the relevant topic listed below for details:

- [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.

Info 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"/>
Next > Finish Cancel	

- If you want to add a user with the default Everyone role, click **Finish** now. Else, click **Next** to define a user role other than the default.
- 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) * **Search**

Name	Actions
TestUser	
admin	

[Add New User](#)

- You can change the user's password, roles or delete using the links associated with it.

Info 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.

i 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 dialog box titled "Enter role details". It has a "Domain" field set to "PRIMARY" with a dropdown arrow. Below it is a "Role Name*" field which is empty. At the bottom are three buttons: "Next >" (disabled), "Finish" (disabled), 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**.

i 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

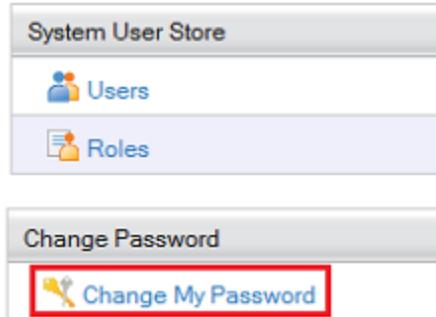
- Some external user stores do not allow you to create empty roles. In that case, selecting users who belong to a role is mandatory.
- If you connect to an external user store in read only mode, you can read existing roles from it but you can not edit/delete the roles. In this case, you can still create 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



3. The **Change Password** page appears. Populate the required fields and click **Change**.

i 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

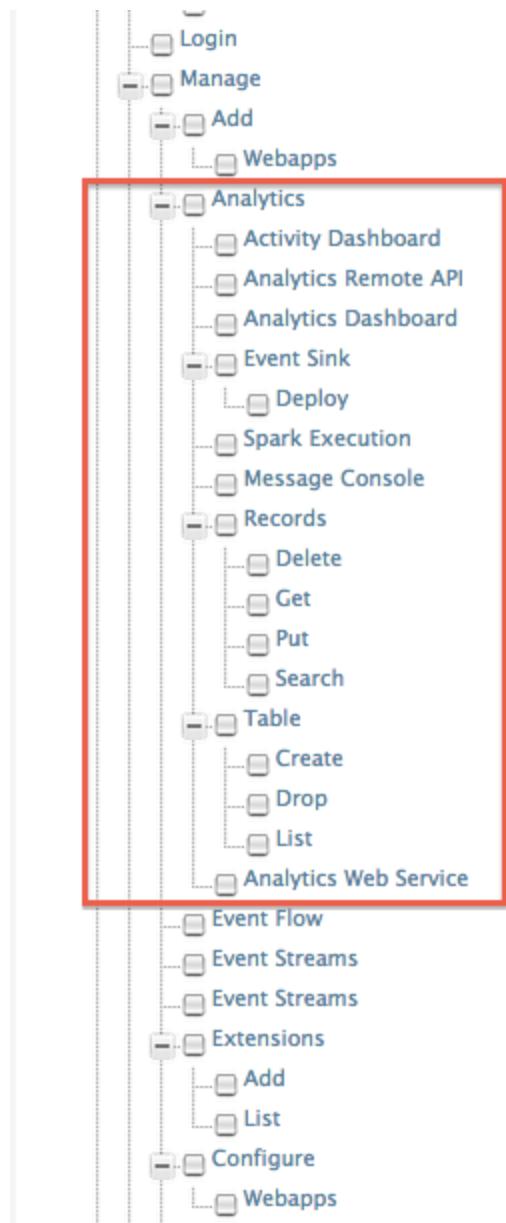
Follow the instructions below to delete a user.

i 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.

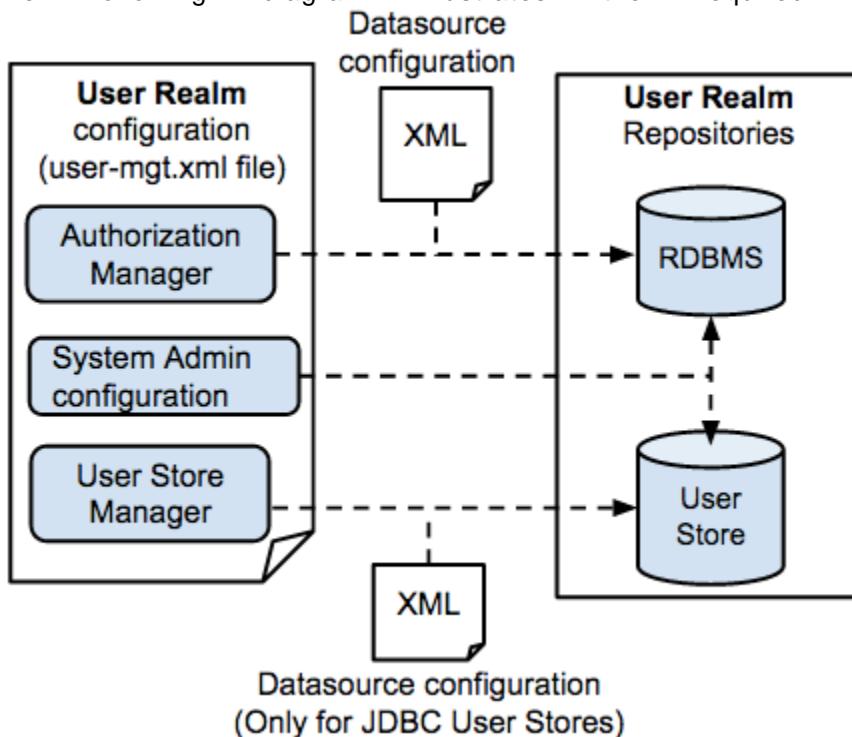
Permission	Description
Activity Dashboard	Required to access and use the activity explorer .
Analytics Remote API	Required to remotely connect to the DAS server, and access the APIs, and also to perform functions of the message console .
Analytics Dashboard	Required to access and use the analytics dashboard . <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <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 message console .
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 message console . 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 message console . 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 message console . 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
<code><AddAdmin></code>	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 <AdminUser> element. If the external user store is in read/write mode, and you set <AddAdmin> to true, 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 AddAdmin value is set to true.
<EveryOneRoleName>	<p>Note: 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.</p>

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
<code>testOnBorrow</code>	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 <code>validationQuery</code> 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 `AuthorizationMa`

nager 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 operati uncommented in the code in
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use ActiveDirectoryUser Domain Service (AD DS) or LDS). This can be used only read-only you must use org APUserStoreManager.
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBCUserStoreMana stores.

The `user-mgt.xml` file already has sample configurations for all of the above user stores. To enable these configurations, you must uncomment them in the code and comment out the ones that you do not need.

The following topics provide details on the various primary user stores you can configure.

- Configuring an external LDAP or Active Directory user store
- Configuring an internal/external JDBC user store

 If you are using `ldaps` (secured) to connect to the Active Directory as shown below, you need to import the certificate of Active Directory to the `client-truststore.jks` of the WSO2 product. See the topic on configuring keystores for information on how to add certificates to the trust-store.

```
<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.

i 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="MaxUserNameListLength">100</Property>
    <Property name="IsEmailUserName">false</Property>
    <Property name="DomainCalculation">default</Property>
        <Property name="PasswordDigest">SHA-256</Property>
    <Property name="StoreSaltedPassword">true</Property>
    <Property name="UserNameUniqueAcrossTenants">false</Property>
    <Property name="PasswordJavaRegEx">[\S]{5,30}\$</Property>
    <Property name="PasswordJavaScriptRegEx">[\S]{5,30}</Property>
    <Property
name="UsernameJavaRegEx">^\[^~!#$;%^*+=\}\\\|\\\\&lt;&gt;,\\\"]{3,30}\$</Property>
    <Property name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="RolenameJavaRegEx">^\[^~!#$;%^*+=\}\\\|\\\\&lt;&gt;,\\\"]{3,30}\$</Property>
    <Property name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
</UserStoreManager>

```

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="userName">shavantha</Property>
    <Property name="password">welcome</Property>
    <Property name="Disabled">false</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="MaxRoleNameListLength">100</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="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_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND
UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID AND
UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = ?</Property>
    <Property name="IsRoleExistingsSQL">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="IsUserExistingsSQL">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.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=?</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>
```

i 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.

i 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/Mess ageDigest.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, 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 access users' attributes and to perform search operations on the user store based on the attribute of the user. This property is mandatory.
ConnectionPassword	Password for the ConnectionName user.
DisplayNameAttribute	This is an optional property. The Display Name Attribute is the name 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 property is used in the search operation only. In this case, the search operation only provides the objects created from the users in the management console.
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 tree.
	<p> Different databases have different search bases.</p>
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 their email address, mobile number, etc.
	<p> The name of the attribute is considered as the username.</p>
UsernameWithEmailJavaScriptRegEx	This property defines the JavaScript regular expression pattern which is used to validate the user name in the configuration file. If you need to support both email as a user name and a password, you can define the regular expression as follows:
	<pre><Property name="UsernameWithEmailJavaScriptRegEx"> <![CDATA[^(?:(?!(^[\w\.-]+@[^\w\.-]+\.[^\w\.-]+)\$)[\w\.-]+ [\w\.-]+@[^\w\.-]+\.[^\w\.-]+)+\$]]> </Property></pre>
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 provide ranges of alphabets, numbers and also ranges of ASCII values.
	<pre><Property name="UsernameJavaRegEx">[a-zA-Z0-9 ._- !#\$%"'*=?^</pre>
RolenameJavaScriptRegEx	The regular expression used by the front-end components for role names.
RolenameJavaRegEx	A regular expression used to validate role names. By default, strings have to provide ranges of alphabets, numbers and also ranges of ASCII values.
ReadGroups	Specifies whether groups should be read from the user store. If this is set to true, then the underlying user store can be read, and 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 be created. 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 this is false, then changes made 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. It is recommended to use a regular expression for defining a UserDNPattern. This provides more impact on performance than using a static DN for all users.
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 or not.
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 JDBC).
PasswordDigest	(JDBC) Digesting algorithm of the password. Has values such as, MD5, SHA1, SHA256, SHA512, 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. By default, strings have to provide ranges of alphabets, numbers and also ranges of ASCII values.

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 tenant specific configuration.
SharedGroupEnabled	This property is used to enable/disable the shared role functionality.
SharedGroupSearchBase	Shared roles are created for other tenants to access under the merged 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:

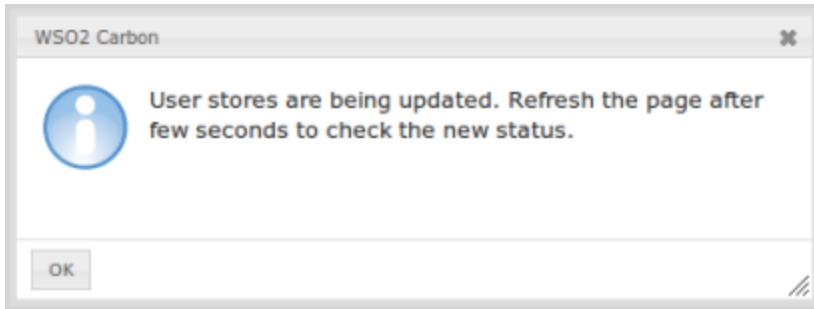
User store manager	Description
org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager	Use ReadOnlyLDAP user stores.
org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Use ReadWriteLDAP read and write operations.
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use ActiveDirectory Domain Service (AD LDS). This can be used in read-only, you must use the LDAPUserStoreManager.
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBCUserStoreManager. It can be configured for properties: <Property

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](#).

- Enter a unique domain name with no underscore (_) characters, and optionally enter a description for this user store.
- 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

- Ensure that all the mandatory fields are filled and a valid domain name is given and click **Add**.
- A message appears saying that the user stores are being added.



i 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 `wso2_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 (`wso2server.bat`) to the product's distribution home (e.g., `-Dcarbon.home=C:\Users\VM\Desktop\wso2datas-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,

Add Repository

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:/userrepo, /home/user/p2-repo</small>	
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

i 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.

- After adding, you can change the repository name using the **Edit** link associated with it.

i You cannot change the repository URL after adding it. To change the URL, you must remove the old repository and add a new one.

- By default, all repositories are enabled. You can disable a repository using the **Disable** link associated with it.

Manage Repositories

Add/edit/remove/disable repositories which contains Features.

Add Repository

Available Repositories:

Name	Location	Enabled	Actions
p1	http://dist.wso2.org/p2/carbon/releases/3.0.0	Enabled	
my_repository	http://dist.wso2.org/p2/carbon/releases/3.2.0	Enabled	

i When you perform a provisioning operation, metadata and artifacts are searched only from the enabled repositories.

Installing features

- In the **Feature Management** page, click **Available Features** tab. Then, select a repository from the drop-down menu.

Feature Management

[Available Features](#) [Installed Features](#) [Installation History](#) [Repository Management](#)

Available Features

Find new features or updates to installed features in available repositories

Repository: Test - <http://dist.wso2.org/p2/carbon/releases/3.2.0> [+ Add Repository](#)

Filter by feature name:

Show only the latest versions Group features by category

[Find Features](#)

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.

Repository: bambooRepo - <http://wso2.org/bamboo/artifact/WSOCARBON-P2REPO/JOB1/build-14> [+ Add Repository](#)

Filter by feature name:

Show only the latest versions Group features by category

[Find Features](#)

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-ManagerServer	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.

9. From the list of features that appear, select the ones you want to add and click **Install**.
10. The **Install Details** page appears. Verify the provided information and click **Next**.
11. Read and accept the terms of license agreement.
12. The installation process starts. It may take a few minutes to download the necessary components.
13. Once the installation process is complete, click **Finish** and restart the server for the changes to take effect.
14. Go to the **Installed Features** tab to browse through the list of installed features.

Turning your product to a back-end/front-end server

15. WSO2 products support back-end, front-end separation where you can manage multiple back-end servers using a single front-end server. You can convert a given product either to a back-end server or to a front-end server by removing the irrelevant features.

For example, if you want to get only a back-end server, you have to uninstall all the front-end features. To do that, select **Front-end** from the drop down menu as follows:

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.

16. Select the features you want to remove and click **Uninstall**.

Unsuccessful feature installation can cause server startup failures. See [Recovering from Unsuccessful Feature Installation](#).

Recovering from Unsuccessful Feature Installation

After installing features, if you encounter server issues or startup failures, you can revert the current configuration by restoring a previous one using either the management console or the command line. The latter is recommended if you cannot start the server.

Use the following steps to check your feature installation history and revert the server back to a previous installation. In this recovery process, some features might get installed and some uninstalled.

- [Restoring using the management console](#)
- [Restoring using the command line](#)

Restoring using the management console

1. Log in to the management console and select **Features** from the **Configure** menu.
2. In the **Feature Management** page, go to the **Installation History** tab.
3. This tab lists the history of provisioning operations performed on the system. For example,

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

- Click on a configuration to view its details. For example,

Feature Management

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.

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.

- 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.

- Start the product with `-D osgiConsole system property`.
- Once the server is started, type the command `osgi> getInstallationHistory`.
- 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
...
```

- 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:

Logging Configuration

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,

Configure Log4J Loggers				
Filter Loggers by <input type="text" value="dataservice"/>	<input type="button" value="Starts With"/>	<input type="button" value="Contains"/>		
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.DataSourceFileLoader	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

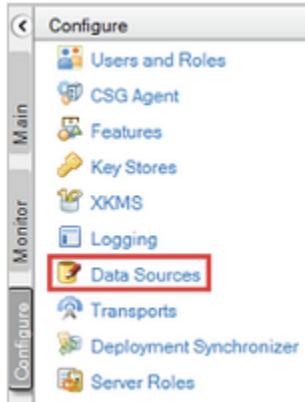
i If datasource management capability is not included in your product by default, add it by installing the above feature. For instructions on the datasource management feature, see [Feature Management](#).

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.

i 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 BAM are defined in the <DAS_HOME>/repository/conf/datasources/ directory as follows.

Datasource	Status	Action
WSO2_ANALYTICS_FS_DB	ACTIVE	View
WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB	ACTIVE	View
WSO2_CARBON_DB	ACTIVE	View
WSO2_ANALYTICS_EVENT_STORE_DB	ACTIVE	View

Master datasources

The `WSO2_CARBON_DB` master datasource, which is defined in the <DAS_HOME>/repository/conf/master-datasources.xml file is used to store data of registry and user manager of WSO2 DAS in a RDBMS.

i 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
<url>	The URL of the database.
<username>	The name of the database user.
<password>	The password of the database user.
<driverClassName>	The class name of the database driver.
<maxActive>	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.
<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.
<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 [DAS Data Access Layer](#).

Datasource Name	Description
WSO2_ANALYTICS_FS_DB	This datasource is used by default to connect to the Analytics File System .
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.

Datasource types

The following sections explain the datasource types, which 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*	<input type="text"/>
Description	<input type="text"/>
Data Source Provider*	default
Driver*	<input type="text"/>
URL*	<input type="text"/>
User Name	<input type="text"/>
Password	<input type="password"/>
<input checked="" type="checkbox"/> Expose as a JNDI Data Source	
<input checked="" 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.XX-bin.jar file to this directory. If you do not copy the driver to this directory when you create the datasource, you will get an exception similar to Cannot load JDBC driver class com.mysql.jdbc.Driver.
- **URL:** The connection URL to pass to the JDBC driver to establish the connection.
- **User Name:** The connection user name that will be passed to the JDBC driver to establish the connection.
- **Password:** The connection password that will be passed to the JDBC driver to establish the connection.
- **Expose as a JNDI Data Source:** Allows you to specify the JNDI datasource.
- **Data Source Configuration Parameters:** Allows you to specify the datasource connection pool parameters when creating a RDBMS datasource.

For more details on datasource configuration parameters, see [ApacheTomcat JDBC Connection Pool guide](#).

After creating datasources, they appear on the **Data Sources** page. You can edit and delete them as needed by clicking **Edit** or **Delete** links.

i When adding an RDBMS datasource, be sure to copy the JDBC driver JAR file for your database to <PROD UCT_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:

New Data Source

New Data Source	
Data Source Type*	RDBMS
Name*	rdbsdatasource
Description	RDBMS Data Source
Data Source Provider*	default
Driver*	com.mysql.jdbc.Driver
URL*	jdbc:mysql://localhost:3306/test
User Name	root
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"/>	

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:

New Data Source

New Data Source			
Data Source Type*	RDBMS		
Name*	rdbmsdatasource		
Description	RDBMS Data Source		
Data Source Provider*	External Data Source		
Data Source Class Name*	<code>ibc.jdbc2.optional.MysqlXADataSource</code>		
<input type="button" value="Add Property"/> <input type="button" value="Delete"/>			
Data Source Properties	Name	Value	Action
	url	:mysql://localhost:3306/test	<input type="button" value="Delete"/>
	user	root	<input type="button" value="Delete"/>
	password	<input type="button" value="Delete"/>	
<input checked="" type="checkbox"/> Expose as a JNDI Data Source <input checked="" type="checkbox"/> Data Source Configuration Parameters			
<input type="button" value="Test Connection"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/>			

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

The screenshot shows the 'New Data Source' configuration dialog. At the top, 'Data Source Type' is set to 'RDBMS'. Below it, various connection parameters are filled: 'Name' (empty), 'Description' (empty), 'Data Source Provider' (set to 'default'), 'Driver' (empty), 'URL' (empty), 'User Name' (set to 'admin'), and 'Password' (set to '*****'). A red box highlights the 'Expose as a JNDI Data Source' section, which includes fields for 'Name' (empty), 'Use Data Source Factory' (with a checked checkbox), and 'JNDI Properties' (with an 'Add Property' button). At the bottom, there are 'Test Connection', 'Save', and 'Cancel' buttons.

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 box. At the top left is the text 'Suspect Timeout'. To its right is a dropdown menu set to 'false'. Below these are three buttons: 'Test Connection', 'Save', and 'Cancel'. The 'Save' button is highlighted.

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	The default TransactionIsolation state of connections created by this pool are as follows: <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)	Used to indicate if objects will be validated before returned to the pool. The default value is false. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> i For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. </div>
Test On Borrow (boolean)	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. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> i 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> . </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 .
	<p> For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.</p>
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 <code>removeAbandonedTimeout</code> . If set to true, a connection is considered abandoned and eligible for removal, if it has been in use longer than the <code>removeAbandonedTimeout</code> . Setting this to true can recover database connections from applications that fail to close a connection. For more information, see 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 <code>setAutoCommit</code> 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 <code>setReadOnly</code> 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)	Connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be <code>[propertyName=property;]*</code> . The default value is null.
	<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>

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.
Alternate User Name Allowed (boolean)	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 username and password, for performance reasons. The pool can however be configured to allow use of different credentials each time a connection is requested. To enable the functionality described in the <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.

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 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_SCHEMA, LOAD ON DATABASE TO USER user
DB20000I  The SQL command completed successfully.
db2 => ■
```



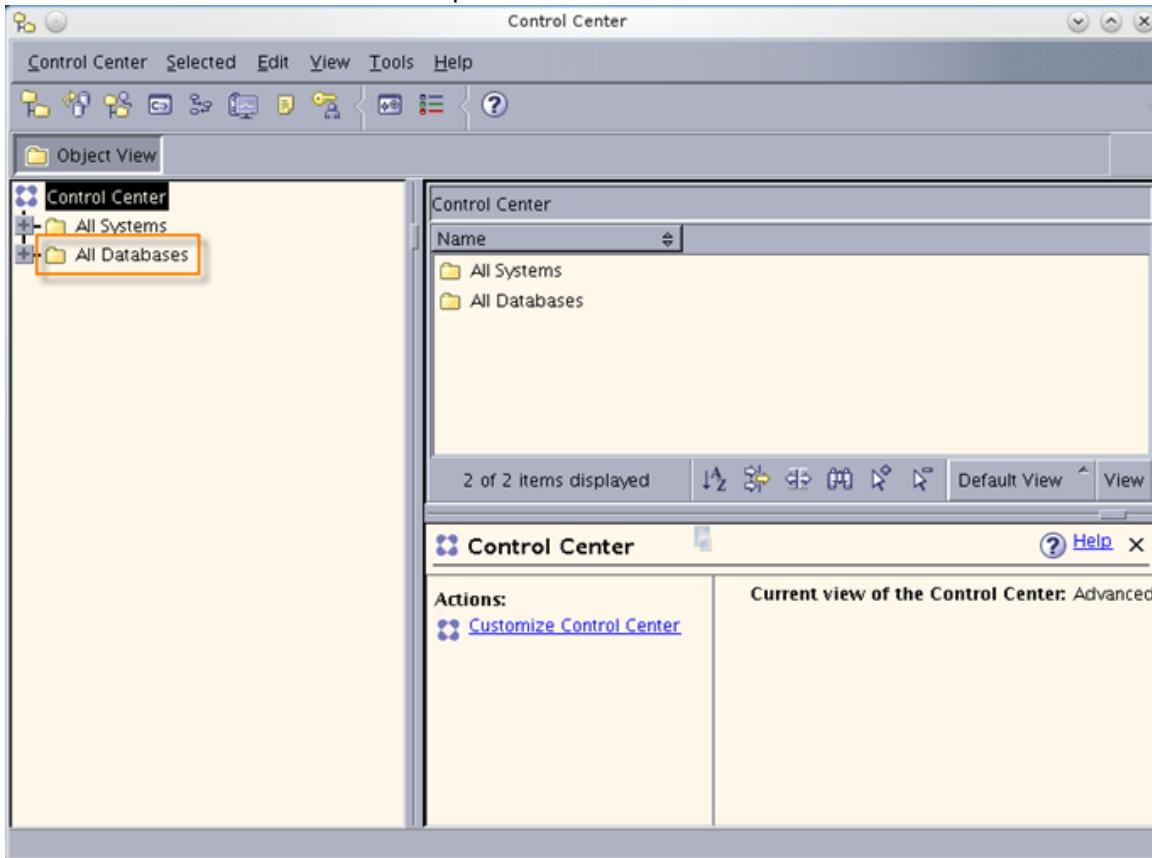
For more information on DB2 commands, see the [DB2 Express-C Guide](#).

Using the DB2 control center

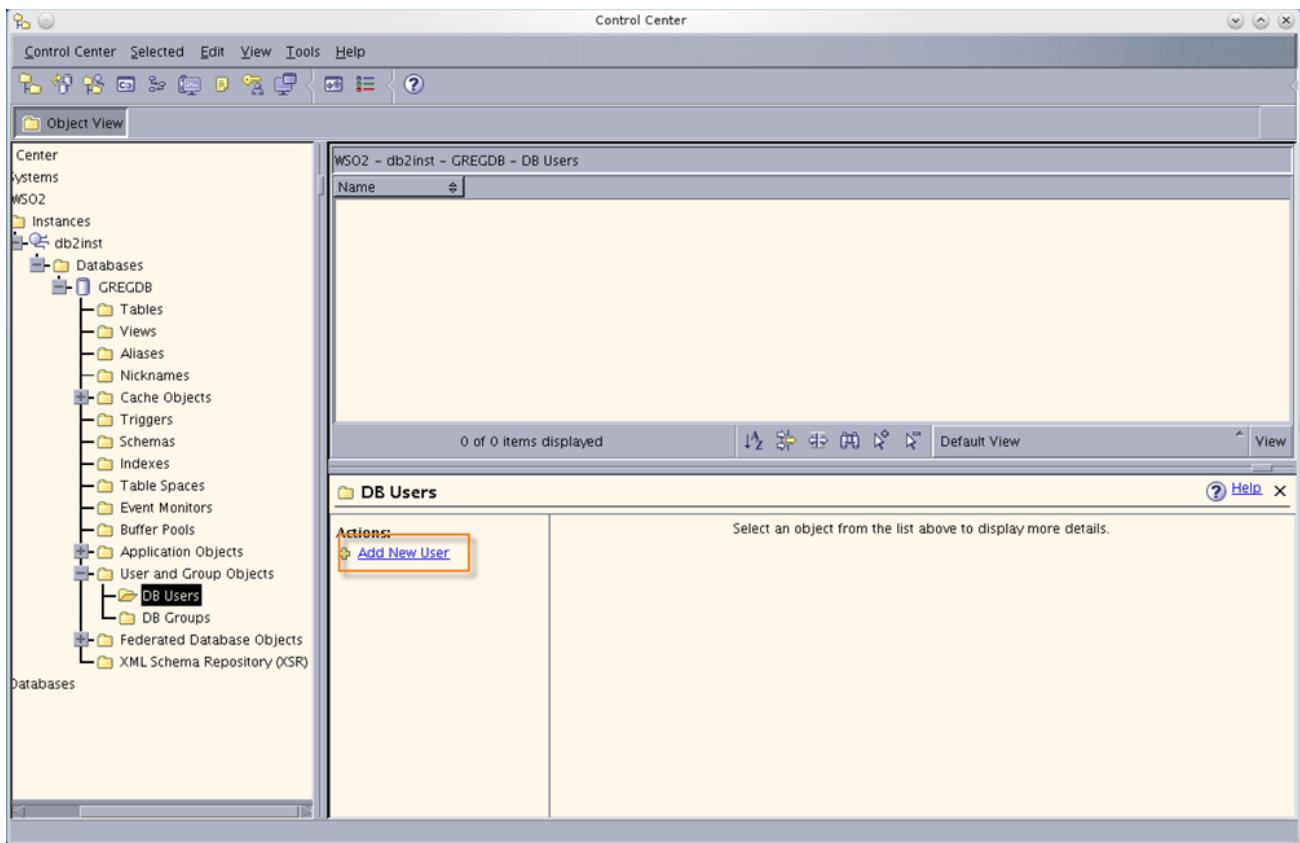
1. Open the DB2 control center using the db2cc command as follows:

```
greg@wso2:~/sqllib/bin$ ./db2cc
```

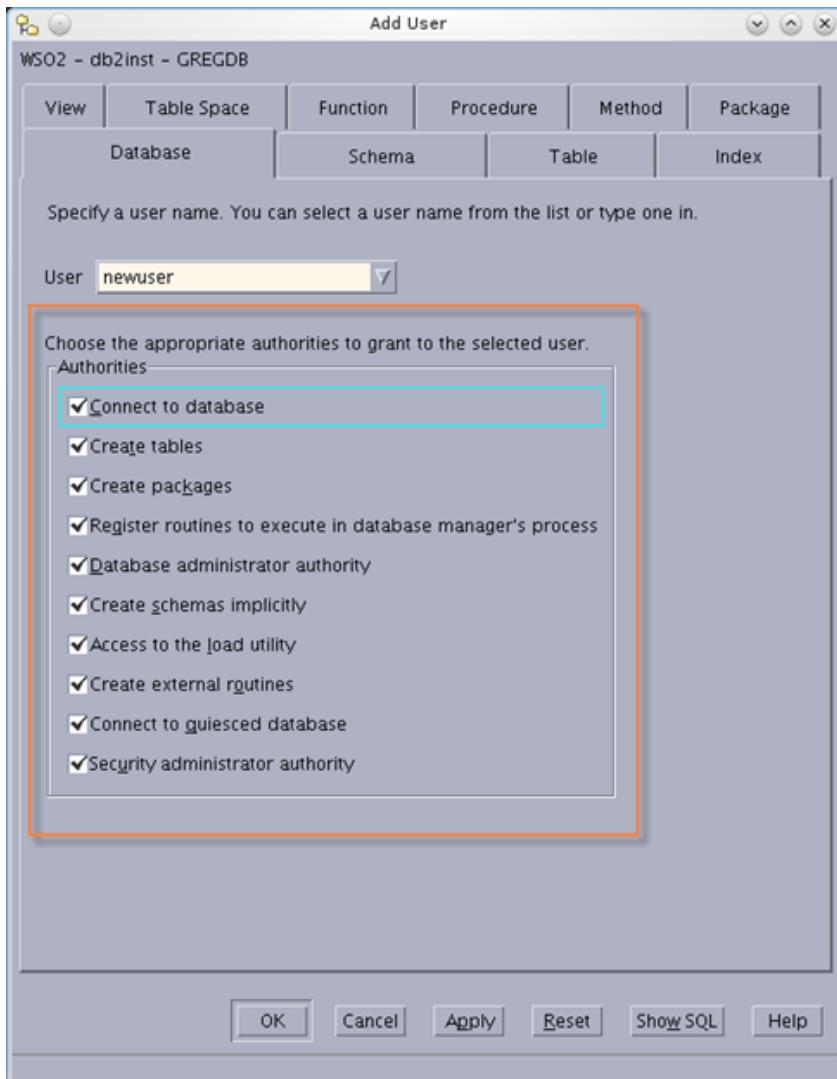
2. Right-click **All Databases** in the control center tree (inside the object browser), click **Create Database**, and then click **Standard** and follow the steps in the **Create New Database** wizard.



3. Click **User and Group Objects** in the control center tree to create users for the newly created database.



4. Give the required permissions to the newly created users.



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:~/sql1lib/java$ cp db2jcc.jar db2jcc_license_cu.jar /home/user/wso2/greg/repository/components/lib/
user@wso2:~/sql1lib/java$
```

i `<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 `<PROD_UCT_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.

(i) 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 `derbynnet.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>

            <driverClassName>org.apache.derby.jdbc.EmbeddedDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<code>url</code>	The URL of the database. The default port for a DB2 instance is 50000.
<code>username</code> and <code>password</code>	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:

1. 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> ij version 10.8
ij> ■
```

2. 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.

3. Exit from the `ij` tool by typing the `exit` command.

```
exit;
```

4. 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';
```

5. Use the scripts given in the following locations to create the database tables:

- To create tables for the **registry and user manager database (WSO2CARBON_DB)**, run the below command:

```
run '<PRODUCT_HOME>/dbscripts/derby.sql';
```

 Now the product is running using the embedded Apache Derby database.

6. Restart the server.

 You can create database tables automatically **when starting the product for the first time** by using the `-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>

        <driverClassName>org.apache.derby.jdbc.ClientDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </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.

i 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](#).

i 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';
```

i 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.

i 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`

i 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.

i 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.

i 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.

i 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
```

i 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

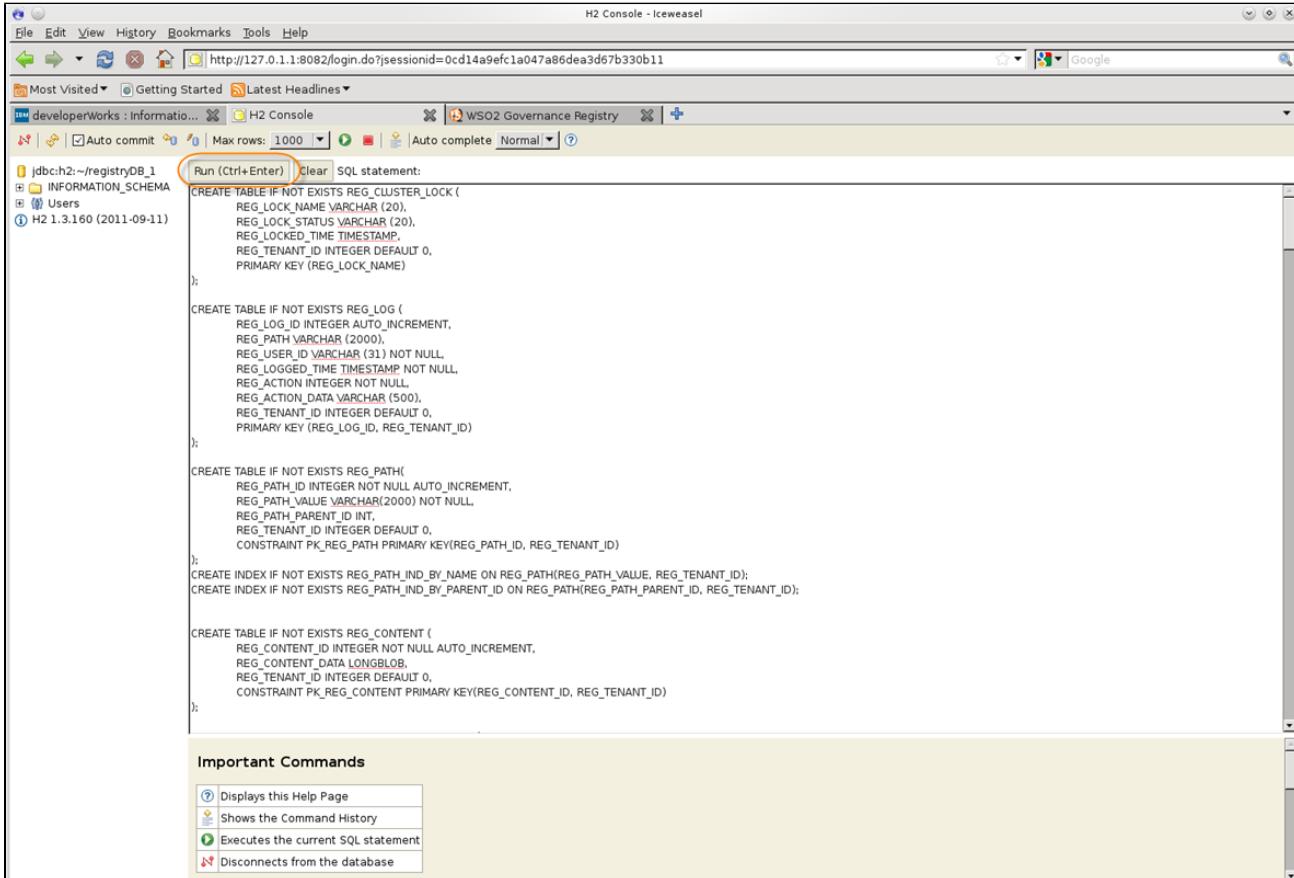
To create the database tables, connect to the database that you created earlier and run the following scripts in H2 shell or web console:

- 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**.



```

H2 Console - Iceweasel
File Edit View History Bookmarks Tools Help
http://127.0.1.1:8082/login.do?sessionid=0cd14a9efc1a047a86dea3d67b330b11
Most Visited Getting Started Latest Headlines
developerWorks : Information... H2 Console WSO2 Governance Registry
Run (Ctrl+Enter) Clear SQL statement:
Max rows: 1000 Auto commit Auto complete Normal
jdbc:h2:~/registryDB_1 INFORMATION_SCHEMA Users H2 1.3.160 (2011-09-11)
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.

i 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>50</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 example above:</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> ✔ Add the following configuration to the connection URL when specifying it as shown 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 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.

i 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.

i 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>50</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.

i 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/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.

i 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:

i 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.

i 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 come 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.s.xml file . Change its elements with your custom values. For instructions, see [Setting up datasource configurations](#).

Setting up MySQL Cluster

Unable to render {include} The included page could not be found.

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 proserv script as follows. Provide the database name and a port as arguments to this script using the -db and -S parameters.

```
proserv -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:opendata://localhost:6767;databaseName=CARBON_DB</url>
                <username>regadmin</username>
                <password>regadmin</password>

            <driverClassName>com.ddtek.jdbc.opendata.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.

i 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.

- i** 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 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:opendata://localhost:6767;databaseName=CARBON_DB</url>
                <username>regadmin</username>
                <password>regadmin</password>

            <driverClassName>com.ddtek.jdbc.opendata.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.

i 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.

- i** 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.

- i** 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.

i 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.

i 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.

i 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.

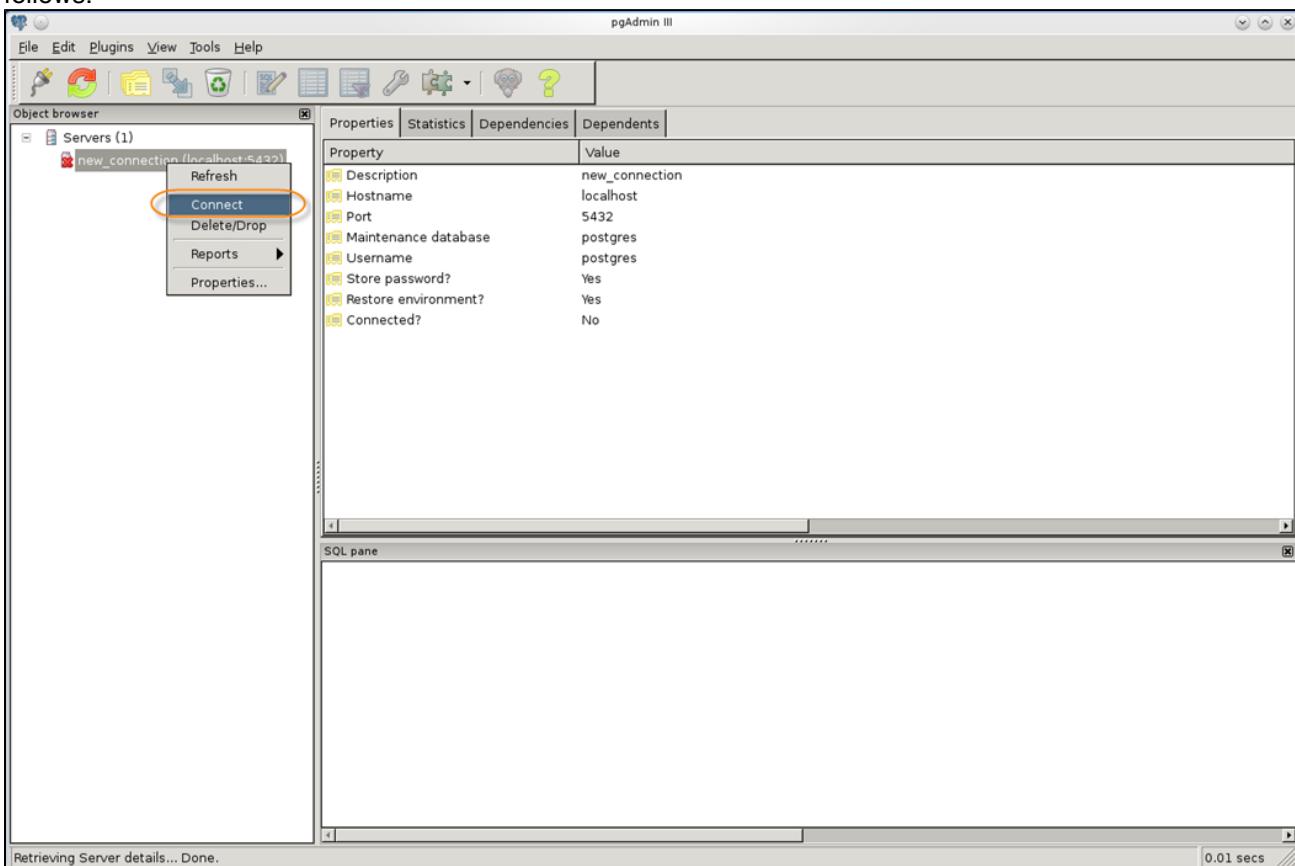
1. Install PostgreSQL on your computer as follows:

```
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>
            <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.

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

Configuring a Cassandra Datasource

Cassandra datasource is used to set up a connection to a Cassandra storage. In WSO2 DAS, you can create a HBase 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>
```

i 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>

```

- i** 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

The screenshot shows the 'New Data Source' configuration page. The 'Data Source Type' is set to 'Custom'. The 'Custom Data Source Type' dropdown is empty. The 'Name' and 'Description' fields are empty. The 'Configuration' section displays a table with two rows, labeled 1 and 2, and includes a toolbar with various icons for configuration.

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.ds
    <customDataSourceProps>
        <property name="inmemory_datasource_schema">{Vehicles:[ID,N
        <property name="inmemory_datasource_records">
            {Vehicles:[{"S10_1678","Harley Davidson Ultimate Choppe
            ["S10_1949","Alpine Renault 1300","Classic Cars","1952"
            ["S10_2016","Moto Guzzi 1100i","Motorcycles","1996"],
            ["S10_4698","Harley-Davidson Eagle Drag Bike","Motorcyc
            ["S10_4757","Alfa Romeo GTA","Classic Cars","1972"],
            ["S10_4962","Lancia A Delta 16V","Classic Cars","1962"],
            ["S12_1099","Ford Mustang","Classic Cars","1968"],
            ["S12_1108","Ferrari Enzo","Classic Cars","2001"]]}
        </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:

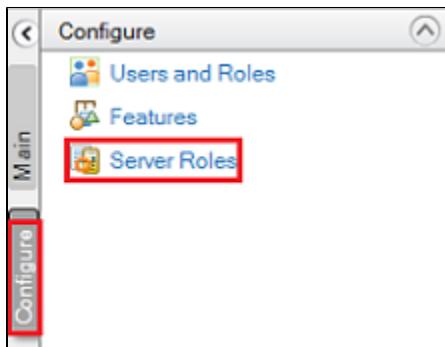
- **Default** - All the server roles picked from that particular product instance's `carbon.xml`.
- **Custom** - All other server roles added by the users.

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*

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.

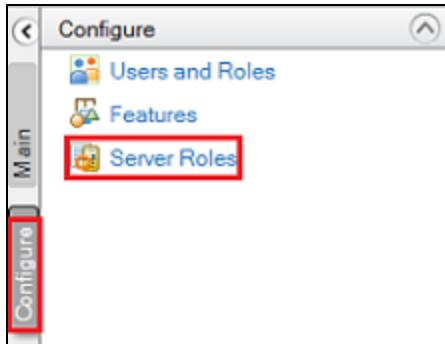
Role Name	Type	Actions
CarbonServer	Default	Delete
TestRole	Custom	Delete

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.



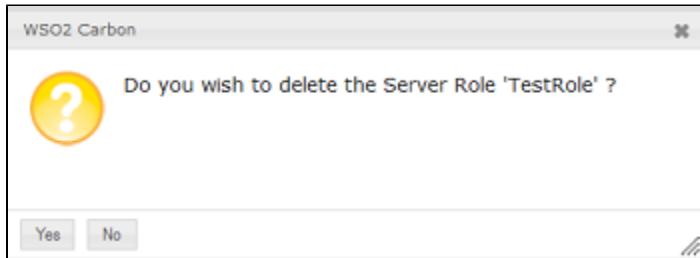
2. The Server Roles page appears. Choose the role you want to delete and click the *Delete* link associated with it.

Server Roles

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.catalina.connector.Connector` implementation from Apache Tomcat. So the servlet HTTP transport actually accepts any parameter accepted by the connector implementation. Please refer to Apache Tomcat's connector configuration reference (<http://tomcat.apache.org/tomcat-5.5-doc/config/http.html>) for more information and a complete list of supported parameters.

Transport sender parameters

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.CommonsHTTPTransportSender` for the receiver and sender in the specified order. The configuration parameters associated with the receiver and the sender are the same as in HTTP transport. This is also a blocking transport implementation.

However, when using the following class as the receiver implementation, we need to specify the servlet HTTPS transport configuration in the transport's XML file.

- `org.wso2.carbon.core.transports.http.HttpsTransportListener`

The class that should be specified as the transport implementation is `org.wso2.carbon.server.transports.http.HttpsTransport`. In addition to the configuration parameters supported by the HTTP servlet transport, HTTPS servlet transport supports the following configuration parameters:

Note: In transport parameter tables, literals displayed in italic mode under the "Possible Values" column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

Parameter Name	Description	Required	Possible Values	Default Value
----------------	-------------	----------	-----------------	---------------

sslProtocol	Transport level security protocol to be used.	No	TLS, SSL	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

i 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
java.naming.factory.initial	JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.	Yes	A
java.naming.provider.url	URL of the JNDI provider.	Yes	A
java.naming.security.principal	JNDI Username.	No	
java.naming.security.credentials	JNDI password.	No	
transport.Transactionality	Desired mode of transactionality.	No	none, transacted
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	AUTO_ACKNOWLEDGE, CLIENT_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE, NO_ACKNOWLEDGE
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, 1.2
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	

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	tr
transport.jms.CacheLevel	JMS resource cache level.	No	ne co
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	N w
transport.jms.ConcurrentConsumers	Number of concurrent threads to be started to consume messages when polling.	No	A to
transport.jms.MaxConcurrentConsumers	Maximum number of concurrent threads to use during polling.	No	A to
transport.jms.IdleTaskLimit	The number of idle runs per thread before it dies out.	No	A
transport.jms.MaxMessagesPerTask	The maximum number of successful message receipts per thread.	No	A to
transport.jms.InitialReconnectDuration	Initial reconnection attempts duration in milliseconds.	No	A
transport.jms.ReconnectProgressFactor	Factor by which the reconnection duration will be increased.	No	A
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	Text, Multi part	Text

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.

i 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.ActiveMQInitialContextFactory</parameter>
        <parameter
name="java.naming.provider.url">tcp://localhost:61616</parameter>
        <parameter
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</parameter>
    </parameter>

    <parameter name="myQueueConnectionFactory">
        <parameter
name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
        <parameter
name="java.naming.provider.url">tcp://localhost:61616</parameter>
        <parameter
name="transport.jms.ConnectionFactoryJNDIName">QueueConnectionFactory</parameter>
    </parameter>

    <parameter name="default">
        <parameter
name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
        <parameter
name="java.naming.provider.url">tcp://localhost:61616</parameter>
        <parameter
name="transport.jms.ConnectionFactoryJNDIName">QueueConnectionFactory</parameter>
    </parameter>
</transportReceiver>
```

<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.

i 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

i 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.

- i** 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 1.6 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.,com.sun.imageio.,org.wso2.carbon. \
-Ddenied.system.properties=javax.net.ssl.trustStore,javax.net.ssl.trustStorePassword,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:	<code>org.wso2.carbon.security.mgt.feature.group</code>				

 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.

3. 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.

1. Execute the following command to generate the CSR:

```
keytool -certreq -alias certalias -file newcertreq.csr -keystore newkeystore.jks
```



As mentioned before, use the same alias that you used during the keystore creation process.

You will be asked to give the keystore password. Once the password is given, the command will output the newcertreq.csr file to the <PRODUCT_HOME>/repository/resources/security/ directory. This is the CSR that you must submit to a CA.

2. You must provide this CSR file to the CA. For testing purposes, try the [90 days trial SSL certificate from Comodo](#).



It is preferable to have a wildcard certificate or multiple domain certificates if you wish to have multiple subdomains like *gateway.sampledomain.org*, *publisher.sampledomain.org*, *identity.sampledomain.org*, etc., for the deployment. For such requirements you must modify the CSR request by adding subject alternative names. Most of the SSL providers give instructions to generate the CSR in such cases.

3. After accepting the request, a signed certificate is provided along with several intermediate certificates (depending on the CA) as a bundle (.zip file).

Sample certificates provided by the CA (Comodo)

The Root certificate of the CA: AddTrustExternalCARoot.crt Intermediate certificates: COMODORSAAddTrustCA.crt , COMODORSADomainValidationSecureServerCA.crt SSL Certificate signed by CA: test_sampleapp_org.crt
--

Step 3: Importing CA-signed certificates to keystore

1. Before importing the CA-signed certificate to the keystore, you must add the root CA certificate and the two intermediate certificates by executing the commands given below. Note that the sample certificates given above are used as examples.

```
keytool -import -v -trustcacerts -alias ExternalCARoot -file AddTrustExternalCARoot.crt -keystore newkeystore.jks -storepass mypassword
keytool -import -v -trustcacerts -alias TrustCA -file COMODORSAAddTrustCA.crt -keystore newkeystore.jks -storepass mypassword
keytool -import -v -trustcacerts -alias SecureServerCA -file 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.

2. After you add the root certificate and all other intermediate certificates, add the CA-signed SSL certificate to the keystore by executing the following command:

```
keytool -import -v -alias <certalias> -file <test_sampleapp_org.crt> -keystore newkeystore.jks -keypass mypassword -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:

1. Get a copy of the `client-truststore.jks` file from the `<PRODUCT_HOME>/repository/resources/security/` directory.
2. Export the public key from your `.jks` file using the following command.

```
keytool -export -alias certalias -keystore newkeystore.jks -file <public key name>.pem
```

3. Import the public key you extracted in the previous step to the `client-truststore.jks` file using the following command.

```
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**.

- 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.

- Log in to the WSO2 product with your user name and password.
- Go to the **Configure** tab and click **Key Stores**.
- 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)		<input type="text"/> <input type="button" value="Search"/>
Name	Type	Actions
wso2carbon.jks	JKS	 Import Cert  View  Delete
 Add New KeyStore		

- 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)						<input type="button" value="Search"/>
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 
entrustssica	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 
equifaxsecureebusinessca2	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 
equifaxsecureebusinessca1	CN=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US	21/06/2020	21/06/1999	4	CN=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:

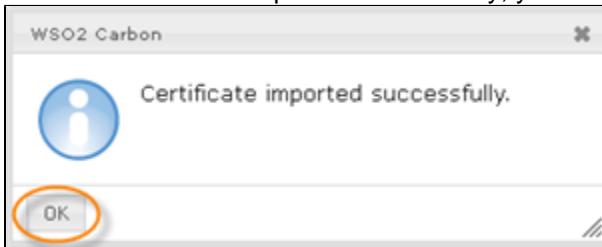
Certificate of the Private Key						
Alias	IssuerDN	NotAfter	NotBefore	SerialNumber	SubjectDN	Version
wso2carbon	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	13/02/2035	19/02/2010	1266562946	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	3
Import Cert Finish						

- 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.

- Log in to the WSO2 product with your user name and password.
- Go to the **Configure** tab and click **Keystores**.
- The **Keystore Management** page appears. All the keystores that are currently added to the product will be listed here as follows:
- Click **Import Cert** associated with the keystore for which you want to import a certificate.
- 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.
- Once you have selected the certificate, click **Import**.
- Once a certificate is imported successfully, you will see the following confirmation:



Click **OK**.

- 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

Installing Machine Learner Features for the CEP Extension

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 Machine Learner features in WSO2 DAS.

Installing the required features in WSO2 DAS

Follow the procedure below to install the required ML features in WSO2 DAS.

- Log into the DAS Management Console following using `admin` as both the user name and the password. For more information, see [Running the Product](#).
- Click **Configure**, and then click **Features**.

3. Click **Repository Management**, and then click **Add Repository**.
4. Enter the details as shown below to add the Carbon P2 repository.

Feature Management

You can add a new local repository or a remote repository

Name: * Carbon P2 repository

Location:

- URL http://product-dist.wso2.com/p2/carbon/releases/4.4.1-SNAPSHOT e.g. http://dist.wso2.org/p2/carbon/releases/4.4.1-SNAPSHOT
- Local e.g. C:\user\repo, /home/user/p2-repo

Add **Cancel**

5. Click **Add**.
6. Click **Available Features** tab, and select the repository added in the previous step.
7. Clear the **Group features by category** check box.
8. Click **Find Features**. It can take a while to list out all the available features in the feature repository. Once listed, select the following features by selecting the relevant check boxes.
 - Machine Learner Core
 - Machine Learner Commons
 - Machine Learner Database Service
 - ML Siddhi Extension

<input type="checkbox"/> Machine Learner	1.0.2	More Info.
<input checked="" type="checkbox"/> Machine Learner Commons	1.0.2	More Info.
<input checked="" type="checkbox"/> Machine Learner Core	1.0.2	More Info.
<input checked="" type="checkbox"/> Machine Learner Database Service	1.0.2	More Info.
<input type="checkbox"/> Machine Learner Jaggery App	1.0.2	More Info.
<input type="checkbox"/> Machine Learner REST APIs	1.0.2	More Info.
<input type="checkbox"/> markdown Module -	1.0.0	More Info.
<input type="checkbox"/> markdown Module -	1.4.0	More Info.
<input type="checkbox"/> Mediation Config Admin	4.4.10	More Info.
<input type="checkbox"/> Mediation Initializer	4.4.10	More Info.
<input type="checkbox"/> Mediation Statistics	4.4.10	More Info.
<input type="checkbox"/> Mediation Tracer	4.4.10	More Info.
<input type="checkbox"/> Mediators	4.4.10	More Info.
<input type="checkbox"/> Message Relay	4.4.10	More Info.
<input checked="" type="checkbox"/> ML Siddhi Extension	1.0.2	More Info.
<input type="checkbox"/> Multitenancy Tenant Management Core	4.4.3	More Info.

If you cannot see these features, retry with one of the following suggestions:

- Try adding a more recent P2 repository. The repository you added could be deprecated.
- Check for the feature in the **Installed Features** tab.

9. Once the features are selected, click **Install** to proceed with the installation.
10. Click **Next**, and then select **I accept the terms of the license agreement**. Then click **Next** again.
11. Once the installation is completed, click **Restart Later**.
12. Stop the CLI server and restart it with the one of tyne following commands. For more information on starting and stopping the server, see [Running the Product](#).

On Windows: <PRODUCT_HOME>\bin\wso2server.bat --run -DdisableMLSparkCtx=true

On Linux/Solaris/Mac OS: sh <PRODUCT_HOME>/bin/wso2server.sh -DdisableMLSparkCtx=true

When you run WSO2 DAS in a distributed mode, the following needs to be carried out.

1. The following dependencies should be uncommented in the <DAS_HOME>/samples/utils/storm-dependencies.jar/pom.xml file as shown below.

```
<!-- Uncomment the following dependency section if you want to include
Siddhi ML extension as part of
Storm dependencies -->

<dependency>
    <groupId>org.wso2.carbon.ml</groupId>
    <artifactId>org.wso2.carbon.ml.siddhi.extension</artifactId>
    <version>${carbon.ml.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.carbon.ml</groupId>
    <artifactId>org.wso2.carbon.ml.core</artifactId>
    <version>${carbon.ml.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.carbon.ml</groupId>
    <artifactId>org.wso2.carbon.ml.database</artifactId>
    <version>${carbon.ml.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.carbon.ml</groupId>
    <artifactId>org.wso2.carbon.ml.commons</artifactId>
    <version>${carbon.ml.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.carbon.metrics</groupId>
    <artifactId>org.wso2.carbon.metrics.manager</artifactId>
    <version>${carbon.metrics.version}</version>
</dependency>

<!--&lt;!&ndash; Dependencies for Spark &ndash;&gt;-->
<dependency>
    <groupId>org.wso2.orbit.org.apache.spark</groupId>
    <artifactId>spark-core_2.10</artifactId>
    <version>${spark.core.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.org.apache.spark</groupId>
    <artifactId>spark-sql_2.10</artifactId>
    <version>${spark.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.org.apache.spark</groupId>
    <artifactId>spark-mllib_2.10</artifactId>
    <version>${spark.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.org.apache.spark</groupId>
    <artifactId>spark-streaming_2.10</artifactId>
    <version>${spark.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.org.scalanlp</groupId>
    <artifactId>breeze_2.10</artifactId>
```

```
    <version>${breeze.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.jblas</groupId>
    <artifactId>jblas</artifactId>
    <version>${jblas.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.spire-math</groupId>
    <artifactId>spire_2.10</artifactId>
    <version>${spire.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>${hadoop.client.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.uncommons.maths</groupId>
    <artifactId>uncommons-maths</artifactId>
    <version>${uncommons.maths.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.json4s</groupId>
    <artifactId>json4s-jackson_2.10</artifactId>
    <version>${json4s.jackson.version}</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.github.fommil.netlib</groupId>
    <artifactId>core</artifactId>
    <version>${fommil.netlib.version}</version>
</dependency>
<dependency>
    <groupId>org.wso2.orbit.sourceforge.f2j</groupId>
    <artifactId>arpack_combined</artifactId>
    <version>${arpack.combined.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
```

```

<artifactId>commons-csv</artifactId>
<version>${commons.csv.version}</version>
</dependency>
```

```

<!-- ML extension dependencies -->

<include>org.wso2.orbit.org.apache.spark:spark-core_2.10
</include>
<include>org.wso2.orbit.org.apache.spark:spark-sql_2.10
</include>
<include>org.wso2.orbit.org.apache.spark:spark-mllib_2.10
</include>
<include>org.wso2.orbit.org.apache.spark:spark-streaming_2.10
</include>
<include>org.wso2.orbit.org.scalanlp:breeze_2.10</include>
<include>org.wso2.orbit.jblas:jblas</include>
<include>org.wso2.orbit.spire-math:spire_2.10</include>
<include>org.wso2.orbit.org.apache.hadoop:hadoop-client
</include>
<include>org.wso2.uncommons.maths:uncommons-maths</include>
<include>org.wso2.json4s:json4s-jackson_2.10</include>
<include>org.slf4j:slf4j-api</include>
<include>org.wso2.orbit.github.fommil.netlib:core</include>
<include>org.wso2.orbit.sourceforge.f2j:arpack_combined
</include>
<include>org.scala-lang:scala-library</include>
<include>org.apache.commons:commons-csv</include>
<include>org.wso2.carbon.ml:org.wso2.carbon.ml.core</include>
<include>org.wso2.carbon.ml:org.wso2.carbon.ml.database
</include>
<include>org.wso2.carbon.ml:org.wso2.carbon.ml.common</include>
<include>
    org.wso2.carbon.ml:org.wso2.carbon.ml.siddhi.extension
</include>
<include>
    org.wso2.carbon.metrics:org.wso2.carbon.metrics.manager
</include>
```

2. Run the following command from the <DAS_HOME>/samples/utils/storm-dependencies-jar directory.

`mvn clean install`

This will generate a jar in the target directory.

Samples

This section demonstrates a few practical examples of the WSO2 Data Analytics Server, their objectives and expected behavior.

- [Sending Notifications Through Published Events Using Spark](#)
- [Analyzing HTTPD Logs](#)
- [Analyzing Smart Home Data](#)
- [Analyzing Realtime Service Statistics](#)
- [Analyzing Wikipedia Data](#)

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 includes 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 [Event Streams](#).

Home > Manage > Event > Streams

Define New Event Stream

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

Event Stream Name*	PRODUCTS_STREAM <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 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

<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
No meta data attributes are defined					
Correlation Data Attributes					
<input type="checkbox"/> Persist Attribute	Attribute Name	Attribute Type	Primary Key	Index Column	Score Param
No correlation data attributes are defined					
Payload Data Attributes					
<input checked="" type="checkbox"/> 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 : <input type="text"/>	Attribute Type : <input type="button" value="INTEGER"/>	Primary Key : <input type="checkbox"/>	Index Column : <input type="checkbox"/>	Score Param : <input type="checkbox"/>	<input type="button" value="Add"/>

[Back](#) [Save Event Stream](#)

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

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"/>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>		

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
<input type="button" value="Configure"/>	

5. Click **Play** in the next screen as shown below.

Send multiple events

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"/>
<input type="button" value="Choose File"/> No file chosen	<input type="button" value="upload"/>		

6. Click **Main**, and then click **Message Console** and select the name of the table to view it as shown below.

Message Console

The screenshot shows the 'Message Console' interface. At the top, there is a search bar with a dropdown set to 'PRODUCTS_STREAM' and a button for 'Schedule Data Purging'. Below the search bar are three radio buttons for searching: 'By Date Range', 'By Primary Key', and 'By Query'. Underneath these are 'Search' and 'Reset' buttons. The main area is titled 'Results' and contains a table for 'PRODUCTS_STREAM'. The table has columns: product-name, quantity, and _timestamp. The data shows four rows: Nike (quantity 7, timestamp 2015-07-20 22:47:26 IST), Adidas (quantity 3, timestamp 2015-07-20 22:47:26 IST), Puma (quantity 2, timestamp 2015-07-20 22:47:26 IST), and Bettans (quantity 0, timestamp 2015-07-20 22:47:26 IST). At the bottom of the table, there are navigation links ('<< < 1 > >>'), a 'Go to page:' dropdown set to 1, a 'Row count:' dropdown set to 25, and a message 'Showing 1-4 of 4'.

PRODUCTS_STREAM			
	product-name	quantity	_timestamp
1	Nike	7	2015-07-20 22:47:26 IST
2	Adidas	3	2015-07-20 22:47:26 IST
3	Puma	2	2015-07-20 22:47:26 IST
4	Bettans	0	2015-07-20 22:47:26 IST

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 <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
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
------------------	----------------------	-----

Action Buttons

[Add Event Stream](#) [Next \[Persist Event\]](#)

5. 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 [WSO2Event Event Receiver](#).

1. Log in to the DAS management console using the following URL: https://<DAS_HOST>:<DAS_PORT>/carbon/
2. Click **Main**, and then click **Receivers**.
3. 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

<p>Event Receiver Name*</p> <p>From</p> <p>Input Event Adapter Type*</p> <p>Usage Tips</p> <p>Adapter Properties</p> <p>Is events duplicated in cluster</p> <p>To</p> <p>Event Stream*</p> <p>Mapping Configuration</p> <p>Message Format*</p> <p>Advanced</p>	<p>SampleEventReceiver Enter a unique name to identify Event Receiver</p> <p>wso2event Select the type of Adapter to receive events</p> <p>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>, ...}</p> <p>For failover: use " " to separate multiple endpoints Eg: {tcp://<hostname>:<port> tcp://<hostname>:<port> ...}</p> <p>For more than one cluster: use "{}" to separate multiple clusters Eg: {tcp://<hostname>:<port> tcp://<hostname>:<port> ...}, {tcp://<hostname>:<port> tcp://<hostname>:<port> ...}</p> <p>Ports available for Thrift protocol - TCP port:7611 or SSL port:7711 Ports available for Binary protocol - TCP port:9611 or SSL port:9711</p> <p>False This depends on how events are published to the server, 'true' only if multiple clusters are present.</p> <p>PRODUCT_ALERTS_STREAM:1.0.0 The event stream that will be generated by the received events</p> <p>wso2event Select the input message format</p>
--	---

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* product_alerts_publisher <small> ⓘ Enter a unique name to identify Event Publisher</small>	From Event Source* PRODUCT_ALERTS_STREAM:1.0.0 <small> ⓘ The stream of events that need to be published</small>
Stream Attributes <small>product-name string, quantity int</small>	
To Output Event Adapter Type* logger <small> ⓘ Select the type of Adapter to publish events</small>	
Dynamic Adapter Properties Unique Identifier <small> ⓘ To uniquely identify a log entry</small>	
Mapping Configuration Message Format* text <small> ⓘ Select the output message format</small>	
Advanced	

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 (receiverURL "tcp://<DAS_HOST>:<DAS_THRIFT_PORT>" ,
         username "<USERNAME>" ,
         password "<PASSWORD>" ,
         streamName "PRODUCT_ALERTS_STREAM" ,
         version "1.0.0" ,
         description "Events are published when product quantity goes beyond a
certain level",
         nickName "product alerts",
         payload "product-name STRING,quantity INT"
) ;

```

 Please provide the appropriate values for DAS_HOST, DAS_THRIFT_PORT, USERNAME and PASSWORD. Default values are, localhost, 7611, admin and admin.

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>
```

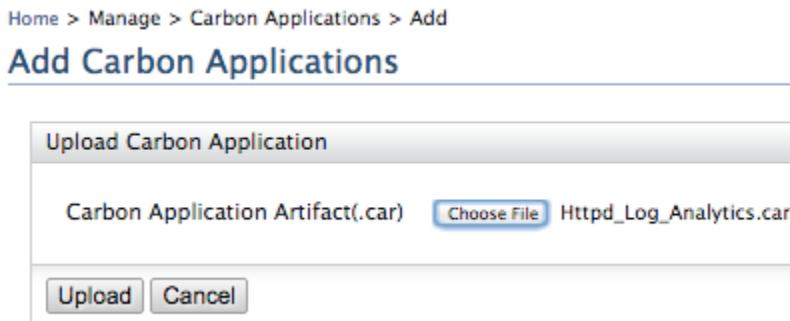
Building the sample

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>/samples/capps/Httpd_Log_Analytics.car file as shown below.



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_HTTPD_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	remoteip	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	

Showing 1-25 of 2108

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

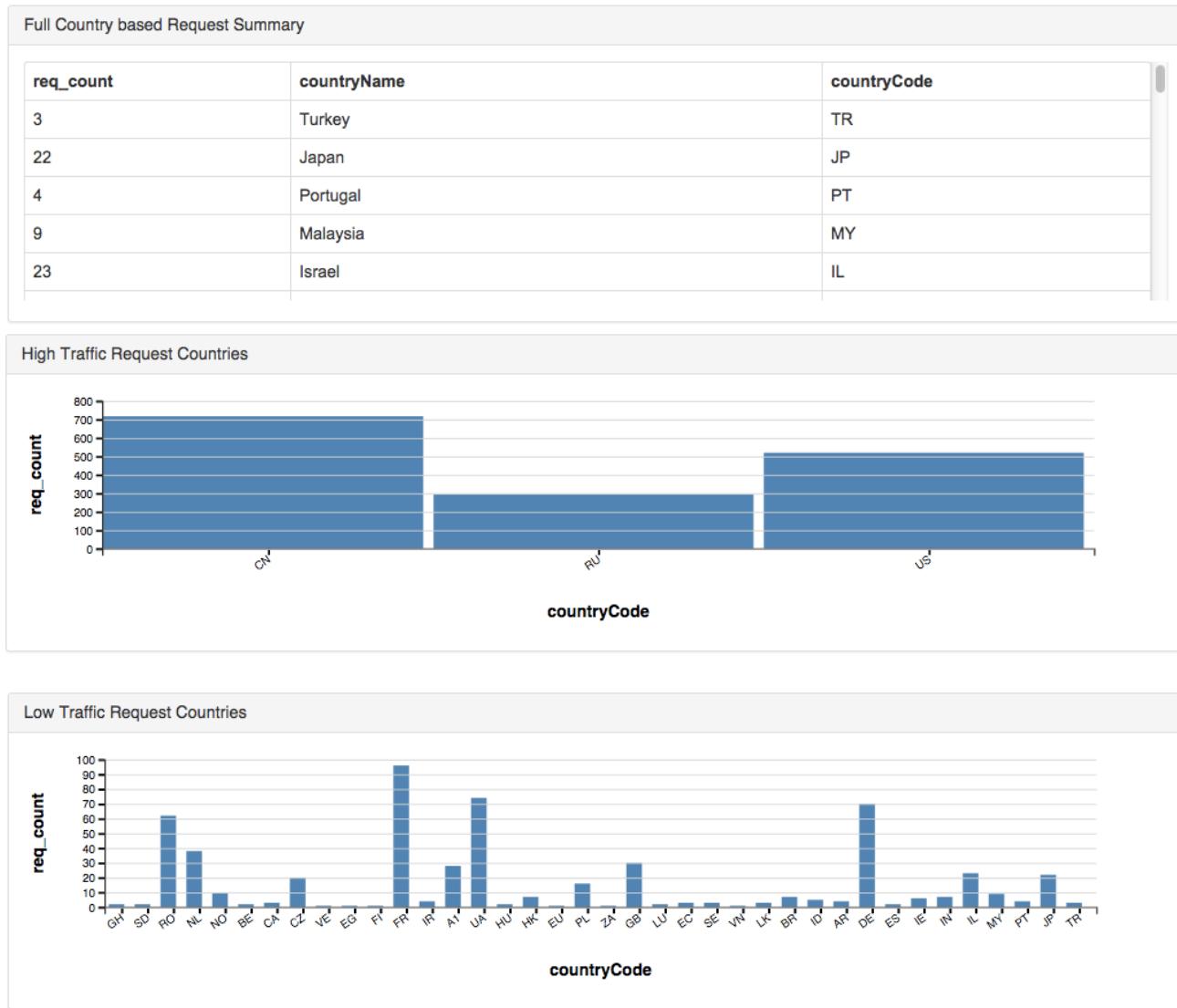
* View, Modify and Delete dashboards

HTTPD Log Analysis Dashboard

This dashboard contains the gadget related to logs analysis based on country.

View
Design
Settings

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 [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>/samples/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

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* Schedule Data Purging

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	529.74426	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

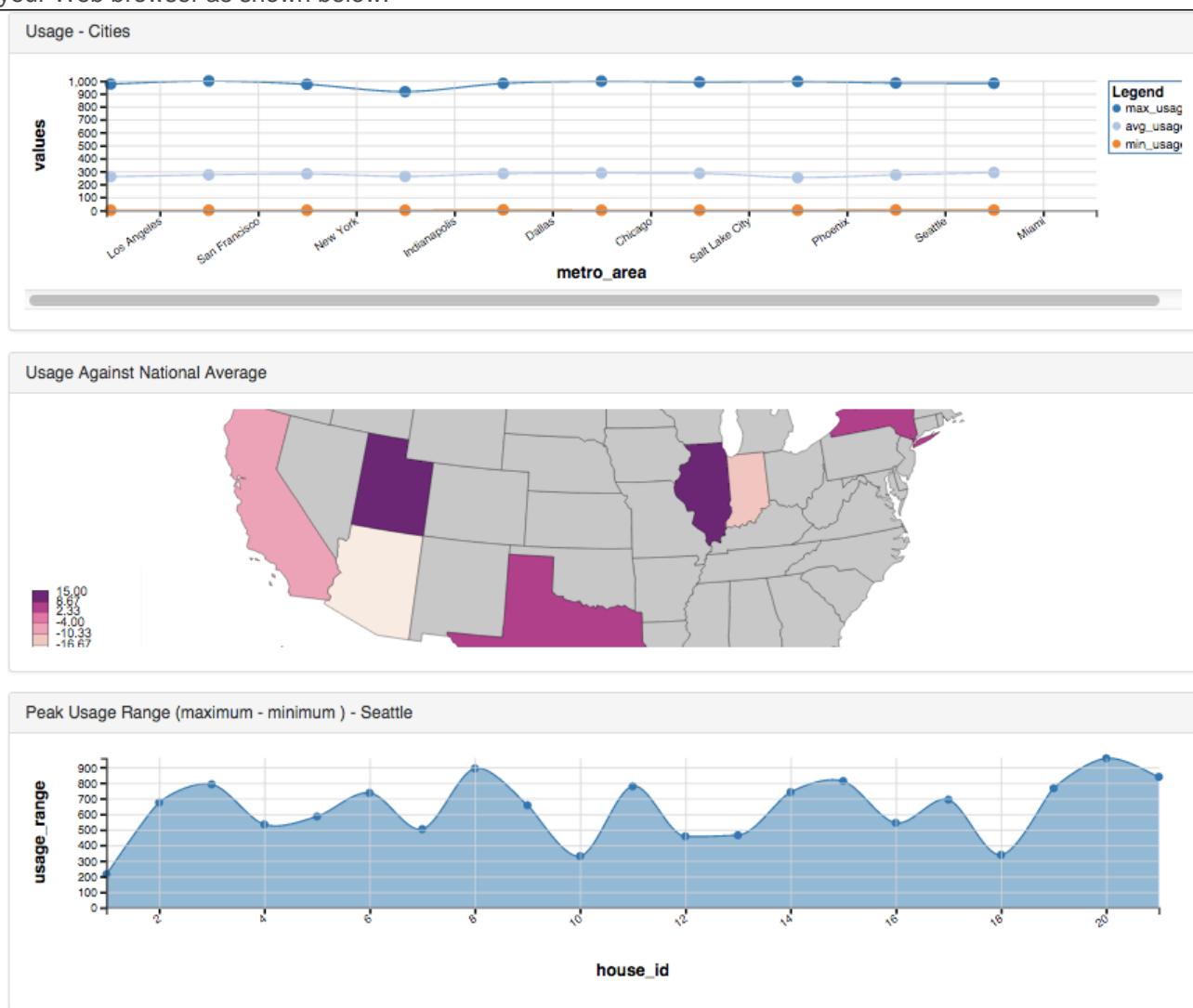
Follow the steps below to use the [Analytics Dashboard](#) to view the output.

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.



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>/samples/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.

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>/samples/capps/Wiki[pedia.car` file as shown below.

Add Carbon Applications

Upload Carbon Application

Carbon Application Artifact(.car) Wikipedia.car

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

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.

- 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>
```

- 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>
```

- 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`

- i Set the values of the `-Dpath` 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 [above 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

Select a Table

Table Name* **ORG_WSO2_DAS_SAMPLE_WIKIPEDIA_DATA**

WIKI_CONTRIBUTOR_SUMMARY
WIKI_AVG_ARTICLE_LENGTH
WIKI_TOTAL_ARTICLE_LENGTH
WIKI_TOTAL_ARTICLE_PAGES

Schedule Data Purging

Search **Reset**

4. 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
4ro7vppa5kmm0olegfjztzcwd0vabw	AccessibleComputing	1414279223000	Paine Ellsworth	9092818	add {{WP:RCAT rcat}}s	wikitext	text/x-wiki	#REDIRECT [[Computer accessibility]] {{Redr move from CamelCase up}}	69	2015-09-23 14:24:00 IST

<< < 1 > >> Go to page: 1 Row count: 50 Showing 1-1 of 1

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. Click the following **CREATE DASHBOARD** button in the top navigational bar to create a new dashboard.



5. 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

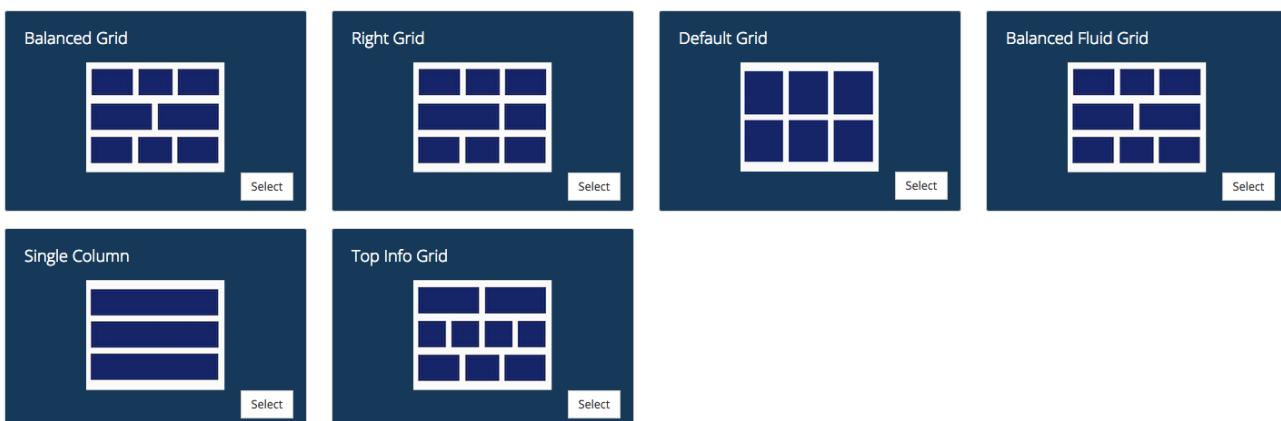
Description

E.g. Monthly Sales Statistics

[Next](#)

- Select a layout to place its components as shown below.

PLEASE SELECT A LAYOUT



- Click **Select** button of the **Single Column** layout. You view a layout editor with the chosen layout blocks marked using dashed lines.
- Click the following **CREATE GADGET** button in the top menu bar.

CREATE GADGET

- 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]

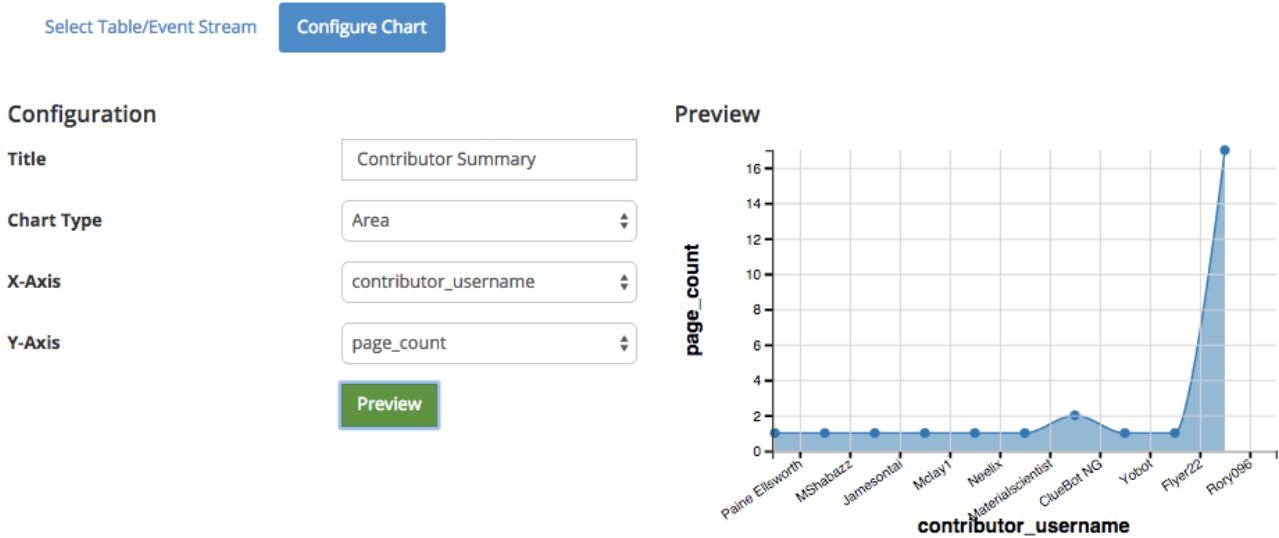
Fields

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**.

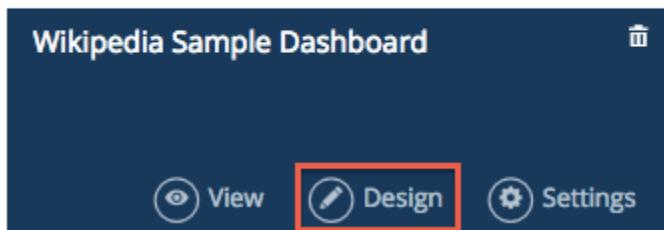
CREATE A GADGET



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

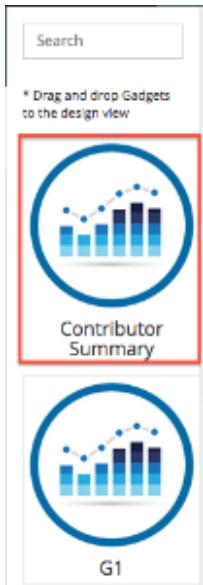
* View, Modify and Delete dashboards



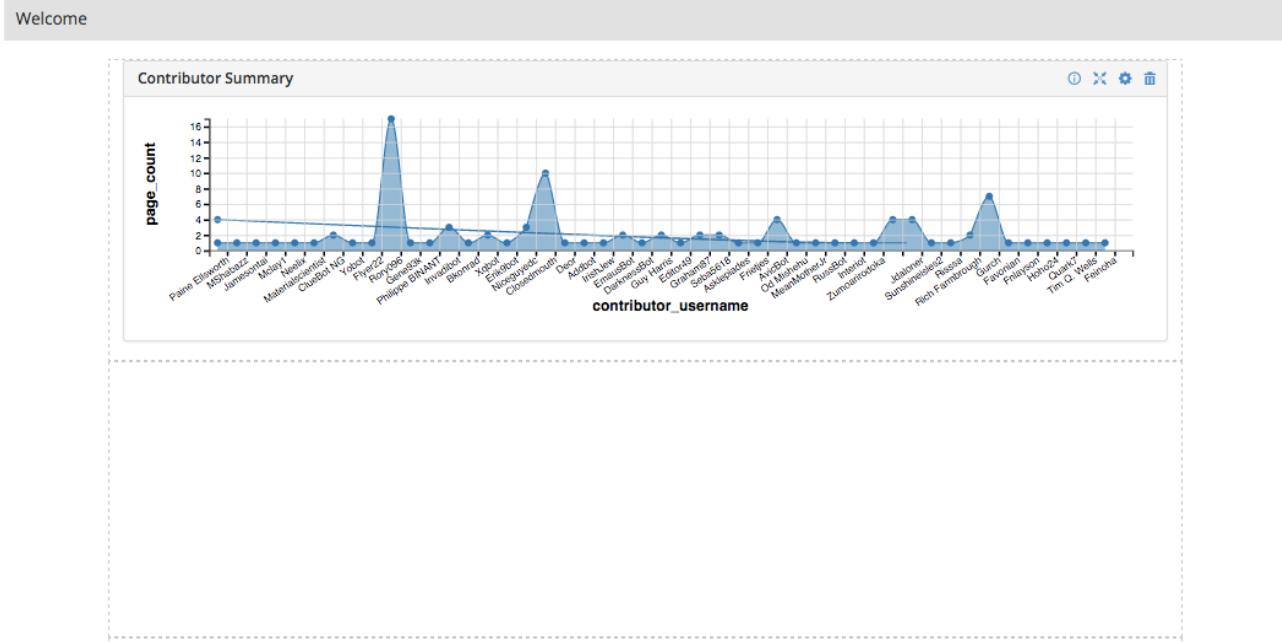
13. Click the following gadget browser icon in the side menu bar.



You view the new gadget listed in the gadget browser as shown below.



- 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.

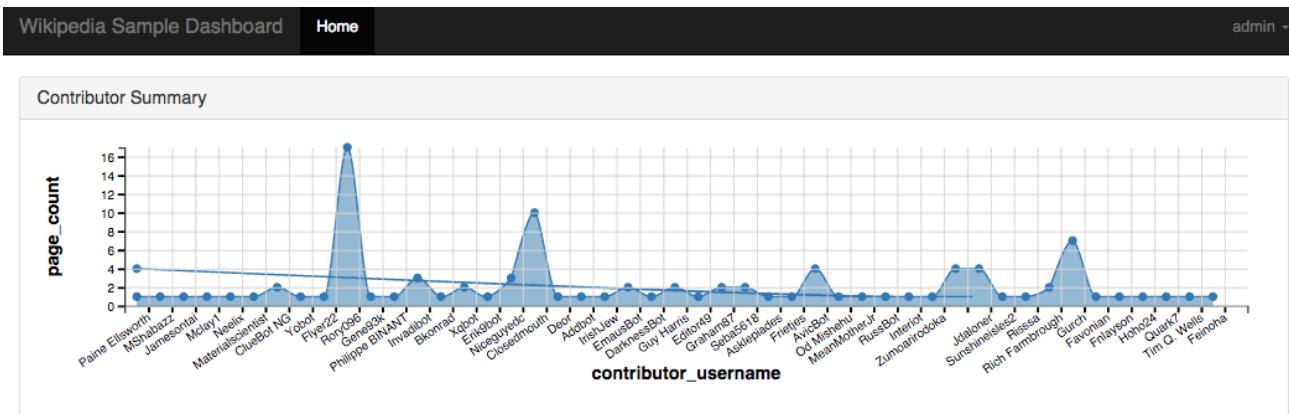


- Click the following **PREVIEW** button in the top menu bar.



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

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 3.0.0.

Function	REST API
Checking if a table exists	GET /analytics/table_exists?tableName={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
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

i 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=

Checking if a Given Table Exists via REST API

- Overview
- Parameter description
- Example
- Sample output
- REST API response

Overview

Description	Check if a particular table exists.
Resource Path	/analytics/table_exists?tableName={tableName}
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic
Username	admin
Password	admin

i 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.

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](#)
- [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

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
- Parameter description
- 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}	Name of the table from which, the records are retrieved
{from}	The starting time to retrieve records from (This is optional.)
{to}	The ending time to get records to (This is optional.)
{start}	The paginated index from value (This is optional.)
{count}	The paginated records count to be read (This is optional.)

Example

```
GET https://localhost:9443/analytics/tables/testtable/1421314718339/1421919518339
```

Sample output

```
[
  {
    "id": "id1",
    "tableName": "table1",
    "timestamp": 1424263913932,
    "values": {
      "columnName1": "value1",
      "columnName2": "value2"
    }
  }
]
```

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](#)
- [Parameter description](#)
- [Example](#)
- [Sample output](#)
- [REST API response](#)

Overview

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

Example

```
GET https://localhost:9443/analytics/tables/testtable/recordcount
```

Sample output

```
1
```

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](#)
- [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

Example

```
POST https://localhost:9443/analytics/search
{
  "tableName": "testtable",
  "query": "product:wso2cdm",
  "start": 0,
  "count": 100
}
```

Sample output

```
[
  {
    "id": "14242656601230.5739142271249453",
    "tableName": "table2",
    "timestamp": 1424265660123,
    "values": {
      "timestamp": "value1",
      "product": "wso2cdm"
    }
  }
]
```

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
- 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

Example

```
POST https://localhost:9443/analytics/search_count
{
  "tableName": "testtable",
  "query": "product:wso2cdm",
  "start": 0,
  "count": 100
}
```

Sample output

```
[
  {
    "id": "14242656601230.5739142271249453",
    "tableName": "testtable",
    "timestamp": 1424265660123,
    "values": {
      "timestamp": "value1",
      "product": "wso2cdm"
    }
  }
]
```

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
- Parameter description
- 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

Example

```
GET https://localhost:9443/analytics/indexing_done?timeout=1
```

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 .
------------------	--

Retrieving the Schema of a Table via REST API

- [Overview](#)
- [Parameter description](#)
- [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}	Name of the table of which, the schema is required

Example

```
GET https://localhost:9443/analytics/tables/testtable/schema
```

Sample output

```
{
    "columns" : {
        "logFile" : {
            "type" : "STRING",
            "isIndex" : true,
            "isScoreParam" : false
        },
        "weight" : {
            "type" : "DOUBLE",
            "isIndex" : true,
            "isScoreParam" : false
        }
    },
    "primaryKeys" : [ "logFile", "logLine" ]
}
```

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
- 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

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
- Example
- Sample output
- REST API response

Overview

Description	Retrieving matching records through a drill-down search
Resource Path	/analytics/drilldown
HTTP Method	POST
Request/Response Format	application/json
Authentication	Basic

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
}
```

Sample output

```
[
  {
    "id": "14242656601230.5739142271249453",
    "tableName": "testtable",
    "timestamp": 1424265660123,
    "values": {
      "timestamp": "value1",
      "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](#)
- [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

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](#)
- [Parameter description](#)
- [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}	Name of the table of which, records are being retrieved using primary key value pairs

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](#)
- [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

Example

```
GET https://localhost:9443/analytics/recordstores
```

Sample output

```
[ "EVENT_STORE" , "PROCESSED_DATA_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
- Parameter description
- 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}	Name of the table of which, the record store is retrieved

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
- Parameter description
- 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}	Name of the record store from which, you need to check if it supports pagination.

Example

```
GET https://localhost:9443/analytics/pagination/EVENT_STORE
```

Sample output

```
{
    "status" : "success" / "non-existent",
    "message" : "EVENT_STORE supported/does not 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
- Parameter description
- Example
- Sample output
- REST API response

Overview

Description	Tracking if the indexing process is complete of a given table.
Resource Path	/analytics/indexing_done?maxWait=10&table={tableName}
HTTP Method	GET
Request/Response Format	application/json
Authentication	Basic

Parameter description

Parameter	Description
{tableName}	Name of the table of which, you need to check if the indexing process is complete.

Example

```
GET https://localhost:9443/analytics/indexing_done?maxWait=10&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
- 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

Example

```
GET https://localhost:9443/analytics/aggregates
{
  "tableName": "test2",
  "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

- Overview
- 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

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 .
------------------	---

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.

i 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.

i 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 - RMIREgistry 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](#) | [Complex Event Processor](#) | [Elastic Load Balancer](#) | [ESB](#) | [Identity Server](#) | [Message Broker](#) | [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)

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.

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 GCM (Google Cloud Message) and APNS (Apple Push Notification Service) and enroll to WSO2 EMM.

A n d r o i d :

The ports to open are 5228, 5229 and 5230. GCM typically only uses 5228, but it sometimes uses 5229 and 5230. GCM 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 the BAM
- 7611 - Thrift TCP port to receive events by the BAM 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.filesystem.port	14000
spark.replClassServer.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.

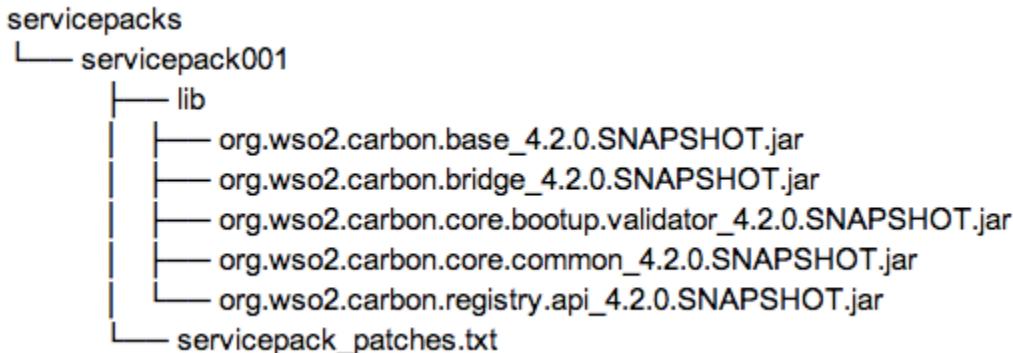
Applying service packs to the product

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.

- i** 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

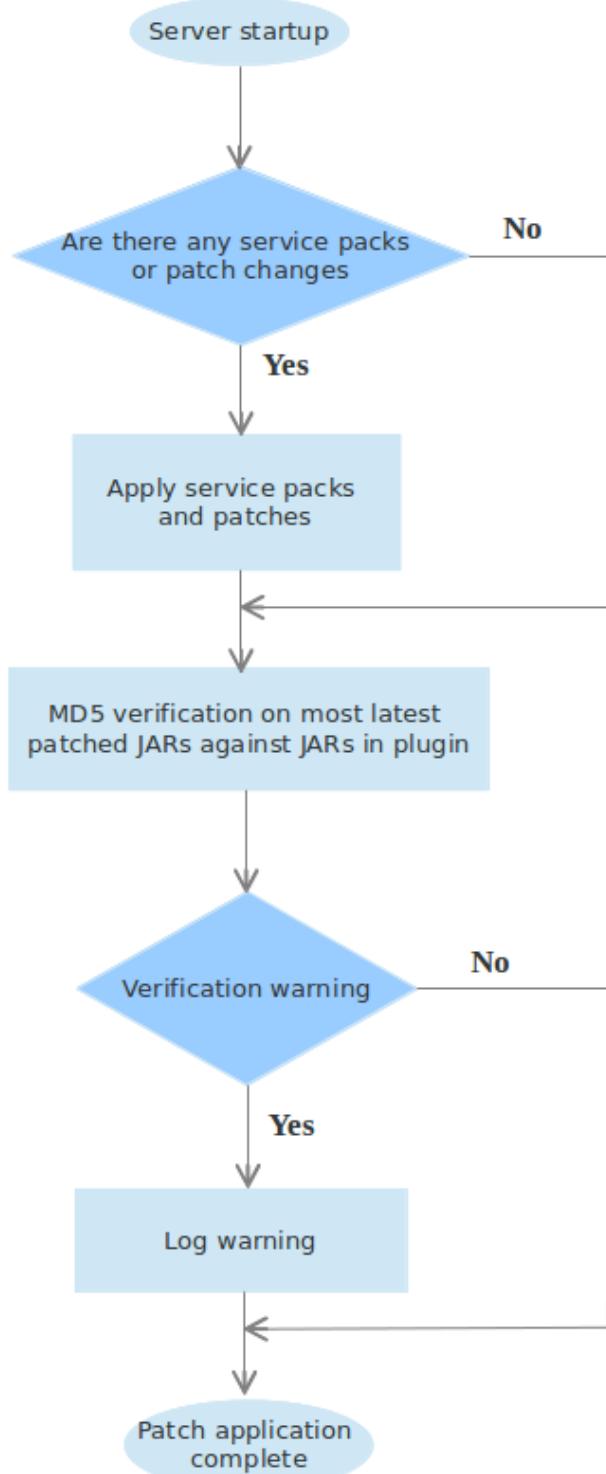
The diagram below shows how the patch application process is implemented when you start the server.

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
- 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 `-D osgiConsole` option, such as `sh <PRODUCT_HOME>/bin/wso2server.sh -D osgiConsole` 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,
2. SynapseApplicationAdmin, SynapseApplicationAdmin, ht
3. CarbonAppUploader, CarbonAppUploader, https://192.16
4. OperationAdmin, OperationAdmin, https://192.168.219.
5. SequenceAdminService, SequenceAdminService, https://
6. MediationLibraryAdminService, MediationLibraryAdminS
7. StatisticsAdmin, StatisticsAdmin, https://192.168.21
8. LoggedUserInfoAdmin, LoggedUserInfoAdmin, https://19
9. MediationStatisticsAdmin, MediationStatisticsAdmin,
10. TopicManagerAdminService, TopicManagerAdminService,
11. MessageProcessorAdminService, MessageProcessorAdmin
12. ApplicationAdmin, ApplicationAdmin, https://192.168
13. NDataSourceAdmin, NDataSourceAdmin, https://192.168
14. ServiceGroupAdmin, ServiceGroupAdmin, https://192.1
15. ClassMediatorAdmin, ClassMediatorAdmin, https://192
  
```

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-name>.jar/web/styles/css/` directory.
3. To specify a new product banner, change the `background-image` attribute of `org.wso2.<product-name>.styles_<version-name>.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-name>.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 | For UNIX

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



Page under construction.

This section summarises the results of performance tests carried out with the following DAS deployments.

- [Standalone DAS node](#)
- [High-availability DAS Setup with two nodes](#)
- [Minimum fully distributed DAS cluster](#)

In each scenario, events were received (both with and without indexing). We have also analyzed events for different samples.

Infrastructure used

- c4.2xlarge Amazon EC2 instances as the DAS nodes
- c3.2xlarge Amazon instance as the database node
- One DAS node was used as the publisher

- c3.2xlarge Amazon instance as database node

Standalone DAS node

 10 million events were published using the [Smart Home sample](#) for this scenario.

Impact of data-bridge configurations

The following parameters in the `<Das_Home>/repository/conf/data-bridge/data-bridge-config.xml` file were assigned different values to test their impact on the throughput.

- `eventBufferSize`
- `BatchSize`

The results were as follows

`eventBufferSize`

Events per Second	<code>eventBufferSize=500</code>	<code>eventBufferSize=1000</code>	<code>eventBufferSize=2000</code>
Average	14641.9899	14641.9899	7555.41414
Median	12487	12264	12021

A significant increase in the average events per second is indicated when the value for the `eventBufferSize` parameter is increased. However, this is caused by queuing and queue flushing fluctuations. The median indicates that a change in the value assigned to the `eventBufferSize` parameter does not have a significant impact on the throughput.

`BatchSize`

Events per Second	<code>BatchSize=100</code>	<code>BatchSize=200</code>
Average	12446.80808	15973.70707
Median	12465	12264

A significant increase in the average events per second is indicated when the value for the `BatchSize` parameter is increased. However, this is caused by queuing and queue flushing fluctuations. The median indicates that a change in the value assigned to the `BatchSize` parameter does not have a significant impact on the throughput.

Spark SQL execution times

The following table shows the Spark execution time for two queries executed on the above sample data. These results are obtained with the default configurations of WSO2 DAS.

Query	Time Taken (seconds)
CREATE TEMPORARY TABLE	0.01 - 0.5
INSERT OVERWRITE TABLE	25 - 40

 Once this query is executed, it will take less time when you execute it again.

High-availability DAS Setup with two nodes

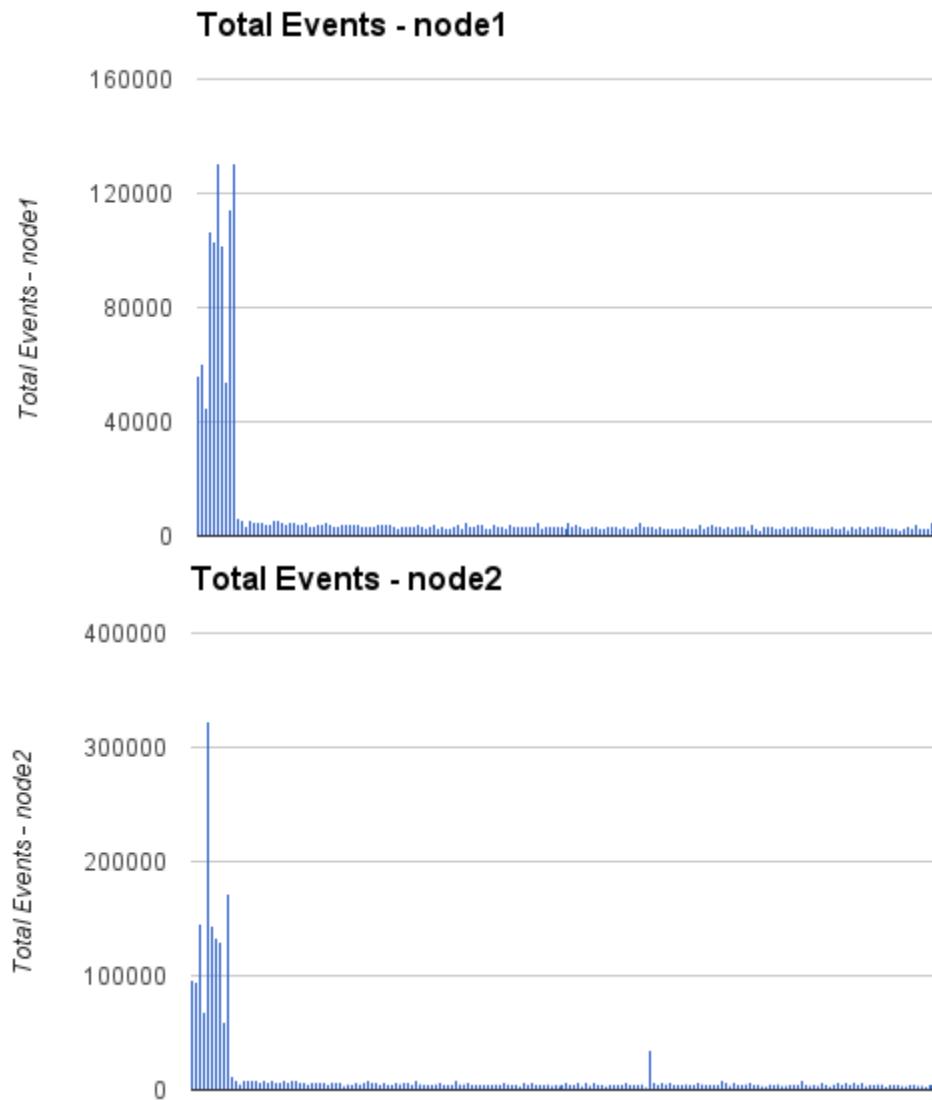
The following table shows a comparison of the receiver throughput of a node in a single node setup, and a node in a two-node HA (High Availability) setup. The test was carried out by publishing 10 million events to WSO2 DAS using the SMART HOME sample.

Events per Second	Single Node	A Node in Two-node HA Setup
Average	15973.70707	15973.70707
Median	12264	12260

The average events per second is significantly higher when the HA setup is used. However, this is caused by queuing and queue flushing fluctuations. The median indicates that the performance of a node in terms of throughput does not significantly change depending on whether it is in a single node setup or a HA DAS setup.

Throughput performance

The following diagrams show the throughput variations in a two-node HA DAS setup. These results were generated from 100 million events published using the [Smart Home sample](#) in batches of 100000 events.

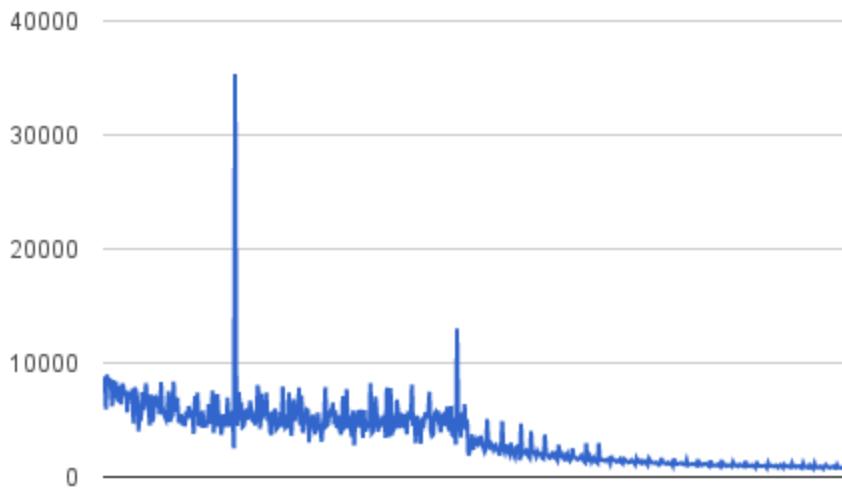


The publishing rate is very high in both nodes at the initial stage. This is because the data receiver can handle a high throughput until the receiver queue is full although the actual event persistence throughput is significantly lesser

than the network bandwidth. The flow control mechanism in the receiver is activated when the receiver queue is full, or when the allocated buffer space is fully used. As a result, event receiving is blocked. This limits and stabilizes the throughput at a level much lower than throughput levels before the receiver queue is full.

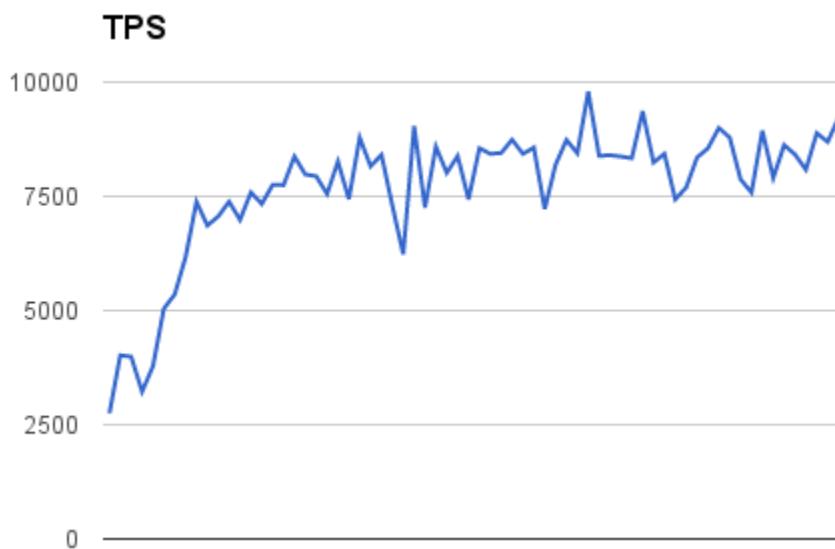
A further reduction in the throughput is observed as follows after 1200000 events. This reduction is caused by limitations of MySQL.

Node 2 throughput from 1200000 onwards

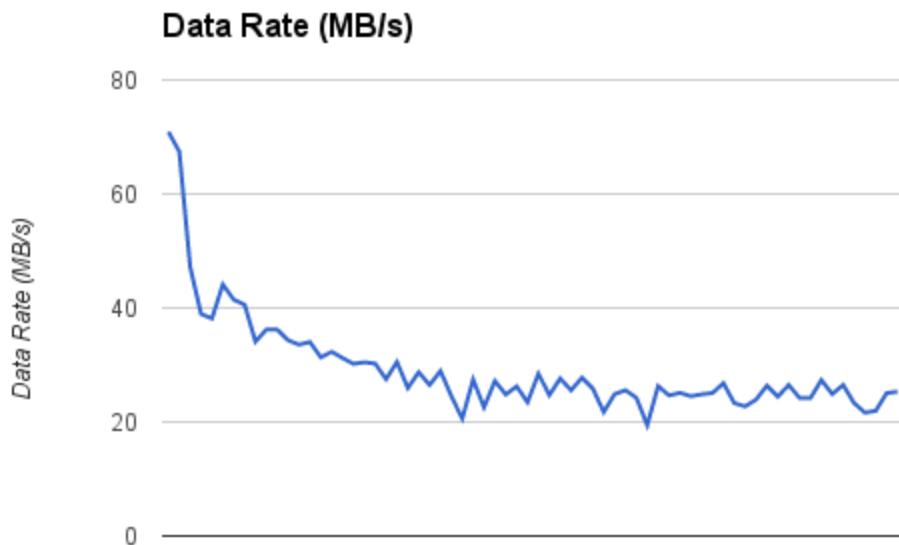


Testing with large events

The following results were obtained by testing the two-node HA DAS cluster with 10 million events published via the [Analysing Wikipedia Data sample](#). Each event in this sample contains several kilobytes, which represents large events.



In the above graph, TPS represents the total number of events published per second. This stabilises 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 stabilises at about 25 MB per second.

Spark SQL execution times

The following table shows the Spark execution time for two queries executed on the 10 million events published via the [Analysing Wikipedia Data](#) sample. These results are obtained with the default configurations of WSO2 DAS.

Query	Time Taken (seconds)
CREATE TEMPORARY TABLE	0.02 - 0.04
INSERT OVERWRITE TABLE	2000 - 7000

Minimum fully distributed DAS cluster

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 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 is built to do batch based analytics on large data volumes. However, it can do real time analytics by installing the WSO2 CEP feature on top of the DAS server. By design, DAS and CEP use the same components to send and receive data, making them compatible to process data. 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)
```

Glossary

[Component](#) | [Port offset](#) | [SOAP](#) | [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.

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:

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

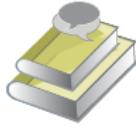
An XML-based, extensible message envelope format, with "bindings" to underlying protocols. The primary protocols are HTTP and HTTPS, although bindings for others, including SMTP and XMPP, have been written.

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 . 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.