# Table of Contents

1. WSO2 Data Services Server Documentation .......................................................... 5
   1.1 About Data Services Server ................................................................................. 5
       1.1.1 Key Concepts and Features .......................................................................... 6
           1.1.1.1 About Data Services and Resources .................................................... 13
       1.1.2 Architecture ................................................................................................. 24
       1.1.3 About this Release ....................................................................................... 25
       1.1.4 FAQ ............................................................................................................. 25
   1.2 Quick Start Guide ................................................................................................. 26
   1.3 Tutorials ............................................................................................................. 35
       1.3.1 Creating a Data Service from Scratch ......................................................... 36
           1.3.1.1 Exposing an RDBMS as a Data Service ............................................... 36
           1.3.1.2 Exposing a Google Spreadsheet as a Data Service .............................. 40
           1.3.1.3 Exposing CSV Data as a Data Service .................................................. 48
           1.3.1.4 Exposing Excel Data as a Data Service ............................................... 52
           1.3.1.5 Exposing RDF Data as a Data Service .................................................. 57
           1.3.1.6 Exposing MongoDB as a Data Service ............................................... 68
           1.3.1.7 Exposing a Web Resource as a Data Service ...................................... 73
           1.3.1.8 Exposing a Carbon Datasource as a Data Service .............................. 77
           1.3.1.9 Exposing a Custom Datasource as a Data Service .............................. 78
           1.3.1.10 Exposing Cassandra as a Data Service ............................................... 84
           1.3.1.11 Exposing a JNDI Datasource as a Data Service .................................. 88
       1.3.2 Uploading a Created Data Service ............................................................... 90
       1.3.3 Generating a Data Service ............................................................................ 90
       1.3.4 Exposing Data as a REST Resource ............................................................. 93
       1.3.5 Invoking Batch Operations .......................................................................... 95
       1.3.6 Invoking an Operation with Multiple Records ............................................ 99
       1.3.7 Defining Nested Queries .............................................................................. 102
       1.3.8 Receiving Notifications from Data Services .............................................. 107
       1.3.9 Handling Distributed Transactions ............................................................. 111
       1.3.10 Exposing Data as an OData Service ........................................................... 115
       1.3.11 Scheduling Tasks ......................................................................................... 116
   1.4 Deep Dive ........................................................................................................... 119
       1.4.1 Installation Guide ......................................................................................... 120
           1.4.1.1 Downloading the Product ................................................................. 120
           1.4.1.2 Installation Prerequisites ...................................................................... 120
           1.4.1.3 Installing the Product .......................................................................... 122
               1.4.1.3.1 Installing on Linux or OS X .......................................................... 123
               1.4.1.3.2 Installing on Solaris .................................................................... 125
               1.4.1.3.3 Installing on Windows .................................................................. 126
               1.4.1.3.4 Installing as a Windows Service ................................................. 129
               1.4.1.3.5 Installing as a Linux Service ....................................................... 136
           1.4.1.4 Running the Product ............................................................................. 138
       1.4.2 Running the product on a Preferred Profile ............................................... 143
       1.4.3 Product Administration ................................................................................ 143
           1.4.3.1 Role-Based Permissions ....................................................................... 147
           1.4.3.2 Upgrading from a Previous Release ................................................... 151
           1.4.3.3 Migrating DSS ..................................................................................... 155
<table>
<thead>
<tr>
<th>Sample Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Processing Sample</td>
<td>268</td>
</tr>
<tr>
<td>Eventing Sample</td>
<td>269</td>
</tr>
<tr>
<td>Secure Data Service Sample</td>
<td>272</td>
</tr>
<tr>
<td>File App Service Sample</td>
<td>274</td>
</tr>
<tr>
<td>XSLT Transformation Sample</td>
<td>276</td>
</tr>
<tr>
<td>Distributed Transactions Sample</td>
<td>277</td>
</tr>
<tr>
<td>Web Resource Sample</td>
<td>279</td>
</tr>
<tr>
<td>REST-Style Resources Sample</td>
<td>280</td>
</tr>
<tr>
<td>JSON Mapping Sample</td>
<td>282</td>
</tr>
<tr>
<td>Cassandra Sample</td>
<td>285</td>
</tr>
<tr>
<td>MongoDB Sample</td>
<td>285</td>
</tr>
<tr>
<td>Custom Datasource Sample</td>
<td>287</td>
</tr>
<tr>
<td>RDF Sample</td>
<td>290</td>
</tr>
<tr>
<td>Expose data as RDF</td>
<td>291</td>
</tr>
<tr>
<td>OData Sample</td>
<td>292</td>
</tr>
</tbody>
</table>
Welcome to the WSO2 Data Services Server 3.5.1 documentation! Data Services Server (DSS) is a shared, elastic, self-service platform for integrating data stores, creating composite data views, and hosting data in different sources such as WS-* style Web services and REST-style Web resources.

**Get started with WSO2 DSS**

If you are new to using WSO2 Data Services Server (WSO2 DSS), follow the steps given below to get started:

- **Get familiar with WSO2 DSS**
  Understand the basics of WSO2 DSS and its architecture.

- **Quick Start Guide**
  Download, install and run WSO2 DSS in just 10 minutes.

**Deep dive into WSO2 Data Service Server**

| Product Administration | Tutorials |

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 Data Services Server

Behind most application silos are heterogeneous data stores. Most businesses require secure and managed data access across these federated data stores, data service transactions, data transformation, and validation. An organization’s data exposed as a service, decoupled from the infrastructure where it is stored is called data services in service-oriented architecture (SOA).

WSO2 Data Services Server facilitates the SOA development efforts by providing a simple, easy-to-use platform for,

- Integrating data stores
- Creating composite data views
- Hosting data in disparate sources such as WS-* style Web services or REST-style Web resources

Data services provide a convenient mechanism to configure a Web service interface for data in various datasources such as relational databases, CSV files, Microsoft Excel sheets, Google spreadsheets etc. These data services provide unprecedented data access and straightforward integration with business processes, mashups, gadgets, business intelligence, and mobile applications. You can expose and access data in a secured (using WS-Security) and reliable (using WS-ReliableMessaging) manner.

The WSO2 Data Services Server is enterprise-ready, lean, 100% open source, cloud-enabled and multi-tenanted. It is released under Apache Software License Version 2.0, one of the most business-friendly licenses available today. It is built on top of the award-winning WSO2 Carbon platform and inherits its cutting-edge middleware capabilities. It also has other features bundled in it by default, such as service management, service dashboard, data service deployment, task scheduling etc.

The topics in this section introduce you to WSO2 Data Services Server, the business cases it solves, features, architecture and how to get help or get involved.

- Key Concepts and Features
- Architecture
- About this Release
- FAQ

Key Concepts and Features

Given below are the key concepts used in WSO2 Data Services Server (WSO2 DSS) and the capabilities or features that are available with the product. If you are already familiar with these concepts and features, move to the tutorials to see how these features work.

[ Create data services using the management console ] [ Expose various datasources ] [ Exposing data as web services/REST resources ] [ Expose data as OData services ] [ Using namespaces for data services ] [ Process requests in batches ] [ Invoke multiple operations simultaneously (boxcarring/request box) ] [ Expose data through various transports ] [ Handling data federation ] [ Manage large data chunks using streaming ] [ Transactional data access and services (distributed transactions) ] [ RESTful data access ] [ Transform and validate data ] [ Secured and managed data access ] [ Monitor data services ]

Create data services using the management console

WSO2 DSS allows you to decouple data (from various datasources) from the infrastructure where it is stored and expose them as data services. You can create a comprehensive data service in a few simple steps using the management console of WSO2 DSS. There are three ways of creating data services in WSO2 DSS.

- Creating a data service from scratch

The management console of WSO2 DSS provides a convenient wizard that will guide you to create the service from scratch using a few simple steps. The following diagram depicts the steps you need to follow.
Let's briefly look at each step of the **Create New Data Service** wizard:

1. **Define the data service:** In the first step, you must specify the general details of the data service, such as the name and description of the data service. In addition, you can enable advanced options for your data service such as **Namespaces**, **Batch Requests**, **Data Streaming**, **Transport settings** and **Distributed Transaction settings**.

2. **Add Datasources:** In the second step, you must specify the details of the datasources that should be exposed by the data service. You can add multiple datasources to the data service to achieve **data federation**. The parameters that you can configure for a datasource depends on the type of datasource. For example, if you use an RDBMS or Cassandra as the datasource, you can **enable OData**.

3. **Define Queries:** In the third step, you must specify queries to process the data in a particular datasource. For example, create queries to search for data in the datasource, or insert/update/delete data. The query you define will consist of parameters that you will use to carry out the query function. For example, to insert data into the datasource, you must specify **input mapping parameters** and to view data from the datasource, you must specify **output mapping parameters**.

4. **Define Operations:** In the fourth step, you can define an operation connected to a single query if you want to **expose the data as a SOAP service**. The operation will be used to invoke the **query**.

5. **Expose data as a REST resource:** In the next step, you have the option of **exposing the data from your data services as REST resources**. To do this, you will be defining a resource connected to each query.

See the following tutorials for step-by-step instructions on how to use the **Create New Data Service** wizard:
- Creating data services from scratch.
- Exposing data as a REST resource.

- **Uploading an already created data service (.dbs file)**

  If you already have a data service created, you can directly upload the .dbs file of the data service using the management console. See **Uploading a Created Data Service** for instructions.

- **Generating a data service using a defined database structure**

  If a database structure is already defined, the data service will be automatically generated for the datasource that you specify. See **Generating a Data Service** for instructions.
Expose various datasources

When you create a data service, you can service enable data from various datasources.

- Supported datasources: Any RDBMS, CSV, Excel, Carbon Datasources, Cassandra, Google Spreadsheets, RDF, JNDI Datasources, Custom Datasources, Any Web page via scraping, MongoDB.
- Supported databases: MSSQL, DB2, Oracle, OpenEdge, TerraData, MySQL, PostgreSQL/EnterpriseDB, H2, Derby or any database with a JDBC driver.

Exposing data as web services/REST resources

Data from various datasources can be exposed as SOAP services or REST-style web resources using WSO2 DSS. When creating a data service, define an operation to expose data as a SOAP service or define a resource to expose data in REST-style.

Expose data as OData services

From DSS 3.5.0 onwards, OData protocol version 4 (OASIS standards) is supported, which mainly provides support for CRUD operations. You can easily expose databases as an odata service by selecting the OData check box in the Create New Data Service wizard. Currently, support for Odata service is available for RDBMS datasources and Cassandra datasources. If you have enabled Odata for your data service, you can complete the data service creation process without defining queries or operations for the service. This Odata service will now be accessible from the following endpoints:

- For super tenant: http://localhost:9763/odata/{dataserviceName}/{datasourceId}/
- For normal tenants: http://localhost:9763/odata/t/{tenantId}/{dataserviceName}/{datasourceId}/

You can find a sample demonstration of an OData service here.

Using namespaces for data services

The service namespace uniquely identifies a Web service and is specified by the <targetNamespace> element in the WSDL that represents the service. A data service is simply a Web service with specialized functionality. As a data service implementation is based on XML, namespace handling is an important aspect to it.

When developing a data service, you get to specify namespaces in several places. One is as shown above. Others are result row namespace, namespaces for specific elements in a query result and for complex results. For more information, see Defining Namespaces.

Process requests in batches

You can enable batch requesting for operations that contain multiple parameters for a single request. When a data service is created with the batch request mode enabled, for all the in-only operations (i.e., operations that do not have any return value), the system creates a corresponding batch operation automatically. This batch operation takes in an array of parameters, compared to the single parameter list a non-batch operation takes.

Batch requests can only be used with in-only operations. That is, the query cannot have a result element or out-type parameters. Batch requests are typically used with insert operations.

For a practical demonstration, see the Batch Processing Sample.
Invoke multiple operations simultaneously (boxcarring/request box)

**Using request_box:** When using the generated wsdl, it will contain all the service level details you need to know in order to invoke a “request_box” request. It’s simply a wrapper element (“request_box”), which wraps the operations you need to call. You can put any number of operations within that “request_box” element any number of times. The order of operations depend on the flow you need to follow. Furthermore, this will return the result of the last operation you have put in the “request_box” wrapper element. The “request_box” operation will be automatically added to the data service.

From DSS 3.5.0 onwards we are deprecating the back end approach used for boxcarring. Instead, we are introducing a new request type (**request_box**), where you can call multiple operations via a single service call. In WSO2 DSS 3.5.0, both boxcarring and request box is disabled in a data service by default. You can enable both these options by clicking the **Enable Boxcarring** check box when you create a new data service. The same "enableBoxcarring="true" attribute in the .dbs file applies to both boxcarring and request box.

**About Boxcarring**

Boxcarring is a method of grouping a set of service calls together and executing them at once. Where applicable, a boxcarring session works in a transactional manner such as when used with an RDBMS datasource. WSO2 Data Services Server facilitates boxcarring by grouping service calls in the server side. As a result, special service clients are not required and as usual, successive service calls can be made to the server to participate in a boxcarring session.

The following control operations get automatically added to the data service when boxcarring is used:

- **begin_boxcar:** A boxcarring session must be started by calling this operation. Once this is called, the server is notified that the subsequent operation calls must belong to this boxcarring session. The server stores these calls without executing them immediately.
- **end_boxcar:** After begin_boxcar is called, the actual operations that belong to the boxcarring session follow. The last step is executing the grouped operations and ending the boxcarring session. This is done with the end_boxcar operation. Once this is called, all the grouped operations are executed at once and the boxcarring session ends. After calling end_boxcar, results are not returned to the client because there are multiple service calls executed at once. If there are any results in the operations of the boxcarring session, they are discarded.
- **abort_boxcar:** If an error occurs in a boxcarring session, the user can choose to end the session by calling the abort_boxcar operation. This invalidates the boxcarring session and removes all pending operations in it.

For boxcarring to function properly, you must use a transport that supports session management, such as HTTP. The service client must also support session management by returning session cookies sent by the server. Boxcarring uses the session information to keep track of operations belonging to one batch. This session information will be sent along the request from the beginning of the boxcarring session to its end. That is, the cookie from the begin_boxcar operation response will be appended to all consequent calls until the boxcarring session is aborted or ended.

**Session management in Axis2 clients**

Axis2 service clients have full support for session management. The following code snippet shows the process of enabling a session in the client and invoking operations in a boxcarring session.
RDBMSSampleStub stub = new RDBMSSampleStub(epr);
stub._getServiceClient().getOptions().setManageSession(true);
stub.begin_boxcar();
stub.addEmployee(49001, "Smith", "John", "john@test.com", 10000.0);
stub.incrementEmployeeSalary(5000.0, 1002);
stub.incrementEmployeeSalary(4500.0, 1003);
stub.end_boxcar();

In line 2 of the code segment above, the client is requested to handle sessions. Without this command, the boxcarring sessions are not created in the server side. The three operation calls that follow the begin_boxcar operation belong to the newly-created boxcarring session. When end_boxcar is called, the earlier three operations are executed together as a group, in one transaction.

**Note:** When using MySQL as the RDBMS, InnoDB Storage Engine must be used.

**Session management in CXF clients**

If the client is CXF, when executing a boxcarring session, you must enable robust in only option with the session management functionality. This is done by setting the endpoint property `org.apache.cxf.oneway.robust` to `true` to activate it at the endpoint. A sample CXF client code is shown below:

```java
private static final QName SERVICE_NAME = new QName("http://ws.wso2.org/dataservice/CXFClientDataService",
CXFClientDataService);
CXFClientDataService ss = new CXFClientDataService(wsdlURL, SERVICE_NAME);
CXFClientDataServicePortType port = ss.getSOAP11Endpoint();
BindingProvider bp = (BindingProvider)port;
bp.getRequestContext().put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
bp.getRequestContext().put(Message.ROBUST_ONEWAY, Boolean.TRUE);
port.beginBoxcar();
```

When you work with boxcarring in a clustered setup, you need to enable sticky sessions in your load balancer. Otherwise, each request will be treated as an individual request, and the transaction behavior will not work as expected for the whole boxcarring session. Note that you do not need sticky sessions when using request box, because request box is always a single request that is sent to a single node in the cluster. The sticky sessions feature is required only when we need to send multiple requests to a single node in the cluster, as in the case of boxcarring.

- See the Request Box Sample for a practical demonstration of the request box.
- See the Boxcarring Sample for a practical demonstration of boxcarring.

**Expose data through various transports**
The transport settings of a data service allow you to specify the transport protocols that should be applicable when receiving messages to the data service. However, this facility to select the transport types as and when you create the data service will directly add the transport settings to the .dbs file of the data service. See the documentation on product administration for information on the transports used in DSS.

Handling data federation

Data from multiple datasources can be exposed through a single data service.

Data federation can be achieved in three ways when you use WSO2 DSS:

- Expose multiple datasources using a single data service.
- Use Nested Queries in your data service. This will allow you to feed the result you get from one query as input to another query. That is, data can be combined into a single response or resource.

For a practical demonstration of how nested queries can be used for data federation, see the Nested Query Sample.

Manage large data chunks using streaming

Data service streaming helps manage large data chunks sent back to the client as the response to a data service query. When streaming is enabled, the data is sent to the client as it is generated, without memory building up in the server. By default, streaming is enabled in data services.

Transactional data access and services (distributed transactions)

Distributed transaction support helps you manage two or more transactions, often using multiple databases, in a coordinated way. A transaction manager is set up in the middle of these transactions for effective coordination and management. This feature uses Java Transaction API (JTA), which allows distributed transactions to be carried out across multiple XA resources in a Java environment. You can also override this transaction manager.

For a practical demonstration of distributed transaction, see Distributed Transactions Sample in Samples.

Distributed transactions with JMS

JMS has the ability to participate in distributed transactions. You can use JMS as the communication channel and run your database transactions in a distributed manner. It guarantees no message loss or failures and ensures single, successful message delivery.

If a JTA transaction is already started, incoming messages will automatically bind with it. In this case, the transaction is started using the JMS transport receiver, if the receiver is enabled as follows:
- Enable the JMS transport receiver in `<PRODUCT_HOME>/repository/conf/axis2.xml` file. The `axis2.xml` file contains properties related to Apache ActiveMQ JMS broker by default. Change the settings as suitable to the JMS broker you use.
- Add dependent libraries to `<PRODUCT_HOME>/repository/components/lib`.
- Add the following two properties to configure the transport receiver to use JTA transactions:
  - `jta`  
  - `true`

After the JMS transport is configured, you can use any JMS client to send messages to the configured JMS broker. For example, you can use the Try-it tool in WSO2 Data Services Server. Be sure to enable the JMS transport sender by un-commenting the code block for JMSSender in `<PRODUCT_HOME>/repository/conf/axis2.xml` file.

**Overriding the default transaction manager**

The default transaction manager of a server can be overridden by providing the location to a JNDI name that the transaction manager is bound to. This is especially useful when the server is embedded in other application servers that do not use the standard JNDI name to bind to the transaction manager. In this case, you can provide a custom JNDI name to detect the transaction manager.

Shown below is an example how WebLogic's JTA transaction manager overrides a data service's default transaction manager.

![Edit Data Service (DTPSampleService)](image)

**RESTful data access**

Data can be exposed as REST resources in addition to SOAP services.

- Create collections of REST resources for CRUD operations.
- Support for both JSON and XML media types.
- Secure resources with HTTP(S) Basic Auth integrated to enterprise identity systems (via WSO2 Identity Server).

For a practical demonstration of this feature, see the sample on REST-Style Resources Sample.
Transform and validate data

- Transform via XSLT and XQuery.
- Built-in validators for standard data types.
- Customizable validators via Java extensions.
- Schema validation via integration with WSO2 Enterprise Service Bus.

For a practical demonstration of this feature, see the XSLT Transformation Sample.

Secured and managed data access

- Fine-grained authorization with XACML (via integration to WSO2 Enterprise Service Bus).
- Comprehensive user management including LDAP via integration to WSO2 Identity Server.

Monitor data services

WSO2 DSS provides a dashboard for monitoring all the running statistics related to data services. You can obtain more advanced monitoring capabilities for your data services by integrating with WSO2 Data Analytics Server (WSO2 DAS).

About Data Services and Resources
Data services and resources provide a service-and-resource-interface to some data stored in a relational database. In a service interface, you must indicate how service requests map to queries against collections of tables in a relational database and how query results are mapped to service responses. In a resource interface, you must indicate how a set of resources map to queries and how query responses are returned as resource representations (or reports of resource creation or deletion, depending on the HTTP verb in use).

The following topics describe the data services configuration language and the key elements used when composing a data service, such as queries, databases, operations etc. along with example syntax.

- Data services and resource language
- Configuring the datasource
- Defining queries
- Defining service operations
- Defining resources
- Defining event trigger
- Security configuration
- Sample data service configuration

### Data services and resource language

Data services and resources are defined using the Data Services and Resource Language (DSRL) where a `<data>` element describes a data service or a resource. The common attributes of a `<data>` element is given in the following example:

```xml
<data baseURI="xs:anyURI" name="xs:NMTOKEN"
enableBatchRequests="xs:BOOLEAN" enableBoxcarring="xs:BOOLEAN"
txManagerJNDIName="xs:NMTOKEN" serviceNamespace="xs:anyURI"
serviceGroup="xs:NMTOKEN" serviceStatus="active|inactive" transports="http https JMS local">
    config+
    query+
    operation+
    resource+
    event-trigger+
</data>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseURI</td>
<td>a REQUIRED URI indicating the base URI for the operations and resources defined within the <code>&lt;data&gt;</code> element.</td>
</tr>
<tr>
<td>name</td>
<td>a REQUIRED name of the data service.</td>
</tr>
<tr>
<td>enableBatchRequest</td>
<td>an OPTIONAL boolean to enable the batch request feature.</td>
</tr>
<tr>
<td>enableBoxcarring</td>
<td>an OPTIONAL boolean to enable the boxcarring feature.</td>
</tr>
<tr>
<td>txManagerJNDIName</td>
<td>an OPTIONAL JNDI name for overriding the standard JNDI location for locating the JTA transaction manager</td>
</tr>
<tr>
<td>serviceNamespace</td>
<td>an OPTIONAL URI to uniquely identify the web service.</td>
</tr>
<tr>
<td>serviceGroup</td>
<td>an OPTIONAL name that is used to categorize data-services in different groups.</td>
</tr>
<tr>
<td>serviceStatus</td>
<td>an OPTIONAL string to enable WIP (specifies weather the data service is deployed or work in progress) support.</td>
</tr>
</tbody>
</table>
transports | an OPTIONAL string to enable the transports required for the data service. The possible values are "http", "https", "JMS" and "local".

### Configuring the datasource

The following sample config gives the common elements used to connect to a datasource:

```xml
<config id="xs:ID">
  <property name="xs:NMTOKEN">xs:urType</property>+</n
</config>
```

- `config/@id`: an OPTIONAL XML ID identifying the config element. If the configuration file has multiple `<config>` elements, then this attribute is required.

The actual set of properties is defined by each type of database connection (e.g., JDBC will have its own standard set).

### Defining queries

A query consists of parameters that describe how to map the result to an XML element. It is similar to a function that maps some parameters to an XML element. A query definition does not indicate how the parameters are acquired. Instead, it just lists the parameters that are needed, assuming that the parameters will be provided. If the query is at a top level (i.e., direct child of `<data>`) then either an operation definition or a resource definition provides the context for the parameters. If the query is nested within a `<result>` element, then the parameter names refer to column names of the result table described in the `<result>` element of the XML.

The following sample config shows the common attributes of a `<query>` element:
<query id="xs:ID" useConfig="xs:ConfigID" returnGeneratedKeys="xs:BOOLEAN">
  <param name="xs:NMTOKEN" sqlType="xs:string" paramType="SCALAR | ARRAY" type="IN | OUT | IN-OUT" ordinal="xs:integer"
  defaultValue="xs:string" />
  ( <validateCustom class="xs:string"/> | <validateLength minimum="xs:integer" maximum="xs:integer"/>
  | <validatePattern pattern="xs:string"/>
  )
</param>
  ( <sql dialect="xs:NMTOKEN">xs:string</sql>+ | <sparql>xs:string</sparql> )
  <properties>
    <property name="xs:NMTOKEN">xs:string</property>
  </properties>
  <result element="xs:QName" rowName="xs:QName"
  defaultValue="xs:string"/>
  (element | attribute | call-query )*
</result>
</query>

<element name="xs:QName" column="xs:NMTOKEN" requiredRoles="xs:NMTOKEN"
  export="xs:NMTOKEN" exportType="SCALAR | ARRAY" xsdType="xs:QName">
  (element | attribute | call-query )*
</element>

<attribute name="xs:QName" column="xs:NMTOKEN" requiredRoles="xs:NMTOKEN"
  export="xs:NMTOKEN" exportType="SCALAR | ARRAY" xsdType="xs:QName"/>

<call-query href="xs:NCName" requiredRoles="xs:NMTOKEN">
  <with-param name="xs:string" query-param="xs:string"/>
</call-query>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Sub-attributes</th>
<th>Sub-attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td></td>
<td></td>
<td>an OPTIONAL XML ID identifying the query. If &lt;query&gt; is a direct child of &lt;data&gt; then this attribute is required.</td>
</tr>
<tr>
<td>useConfig</td>
<td></td>
<td></td>
<td>a REQUIRED reference to the datasource that is to be used for query.</td>
</tr>
<tr>
<td>returnGeneratedKeys</td>
<td></td>
<td></td>
<td>an OPTIONAL boolean parameter to enable returnGeneratedKeys. Set this attribute to true only in INSERT queries, where the query inserts to a table that has an auto incrementing key column. In such a case, an auto incremented key value is added to the results set. Also see Returning Generated Keys.</td>
</tr>
<tr>
<td>param</td>
<td>a declaration of a parameter of the query</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>a REQUIRED name of the parameter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sqlType</td>
<td>an OPTIONAL string containing a legal SQL type which defines the type of the parameter. If none is specified then defaults to string.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>paramType</td>
<td>a REQUIRED parameter type. If none is specified then defaults to SCALAR.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ordinal</td>
<td>a REQUIRED only for stored procedures which map the parameter positions with the query.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>defaultValue</td>
<td>an OPTIONAL default value of the input parameter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>validateCustom class</td>
<td>a REQUIRED custom validation class to validate the input parameter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>validateLength minimum</td>
<td>a REQUIRED integer when specifying the minimum length of the parameter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>maximum</td>
<td>a REQUIRED integer when specifying the maximum length of the parameter.</td>
<td></td>
</tr>
<tr>
<td>validatePattern pattern</td>
<td>a REQUIRED string pattern to validate the string input parameter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sql</td>
<td>a REQUIRED string containing the SQL query or SQL function to execute. See Calling an SQL Function in a Query.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dialect</td>
<td>an OPTIONAL string containing jdbc driver prefix when need to use sql-dialects.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sparql</td>
<td>a REQUIRED string containing the sparql query to execute when using RDF as datasource.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>an OPTIONAL XML to define advanced query properties. Each property is defined as a child element of this.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>a REQUIRED name of the property.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result</td>
<td>a REQUIRED element describing how the table resulting from executing the query will be converted to an XML element. If any <code>&lt;column&gt;</code> or <code>&lt;query&gt;</code> child elements are present, then ONLY those are transferred as child elements of the result element (or elements, depending on whether result/@rowName is given or not). The order of the nested <code>&lt;column&gt;</code> or <code>&lt;query&gt;</code> elements defines the order of elements in the result element.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>element</td>
<td>a REQUIRED QName which is the name of the element which will hold the results.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rowName</td>
<td>an OPTIONAL QName which is the name of the element wrapping each row of the result table if more than one element from the table is to be returned. If this attribute is not given, then only the first row is returned and hence no second level wrapper element is needed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>defaultNamespace</td>
<td>an OPTIONAL URI being the default namespace to use for the namespace name of elements and attributes that result columns are mapped to. Defaults to &quot;&quot; (meaning no namespace).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>call-query</td>
<td>an OPTIONAL element (which may occur any number of times) which is used to execute a further query and produce an element which will be present in the parent element as a child. This is used primarily to use a value of a column as key to select data from a new table.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>element</td>
<td>an OPTIONAL element (which may occur any number of times) indicating how a specific column in the result table is to be mapped into an element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>element/@name</td>
<td>a REQUIRED QName giving the name of the element to put the column data into</td>
<td></td>
<td></td>
</tr>
<tr>
<td>element/@column</td>
<td>an OPTIONAL string giving the name of the column whose value is to be copied into the element.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>element/@requiredRoles</td>
<td>an OPTIONAL string giving the names of roles that who has permission to see the result element. By default it has set to all users.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element/Attribute</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>element/@export</td>
<td>an OPTIONAL name giving to the element that to be export outside of query. This feature is used with boxcarring support.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>element/@exportType</td>
<td>a REQUIRED parameter when using export option. Used to give the export element type whether scalar or array.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>element/@xsdType</td>
<td>an OPTIONAL indication of the XML Schema type of the element. If none is given defaults to the mapping of the SQL type of the result column named by @column to an XML Schema type as per [SQL XML Mapping]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute</td>
<td>an OPTIONAL element (which may occur any number of times) indicating how a specific column in the result table is to be mapped into an attribute of the element representing the current row</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute/@name</td>
<td>a REQUIRED QName giving the name of the attribute to put the column data into</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute/@column</td>
<td>an OPTIONAL string giving the name of the column whose value is to be copied into the attribute. Either @column or @param is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute/@param</td>
<td>an OPTIONAL string giving the name of the param whose value is to be copied into the attribute. Either @column or @param is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute/@requiredRoles</td>
<td>an OPTIONAL string giving the names of roles that who has permission to see the result attribute. By default it has set to all users.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute/@export</td>
<td>an OPTIONAL name giving to the attribute that to be export outside of query. This feature is used with boxcarring support.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute/@exportType</td>
<td>a REQUIRED parameter when using export option. Used to give the export element type whether scalar or array.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribute/@xsdType</td>
<td>an OPTIONAL indication of the XML Schema type of the attribute. If none is given defaults to the mapping of the SQL type of the result column named by @column to an XML Schema type as per [SQL XML Mapping]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
call-query

| with-param/@name | an REQUIRED name of the query to put the column data into |
| with-param/@query-param | an OPTIONAL string giving the name of the column whose value is to be copied into the element |

**Defining service operations**

Operation refers to a Web service operation defined by a query. The operation is defined as an invocation of a query indicating how the parameters of the query are computed or derived. The syntax is as follows:

```xml
<operation name="xs:NCName" disableStreaming="xs:BOOLEAN"/>
  <description>"xs:string"</description>
  <call-query href="xs:IDREF" />  
    <with-param name="xs:NMTOKEN" (query-param="xs:NMTOKEN" |
    column="xs:NMTOKEN | param="xs:NMTOKEN") />
  </call-query>
</operation>
```

- operation/@name: is the REQUIRED name of the operation.
- operation/@disableStreaming: an OPTIONAL boolean that used to disable streaming. By default streaming are enable.
- operation/@description: an OPTIONAL string used to describe operation.
- operation/call-query: describes how a query is to be invoked with the data received in the operation invocation.
  - call-query/@href: an OPTIONAL reference to the query that is to be invoked. If this is missing then a query must be nested within this element.
  - call-query/with-param: a description of a parameter binding for the query invocation: says how a named parameter's value is computed.
    - with-param/@name: a REQUIRED NMTOKEN identifying the parameter whose value is being specified.
    - with-param/@query-param: an OPTIONAL attribute indicating the name of the URI query parameter (from operation/@path) whose value is the value of this parameter.
    - with-param/@column: an OPTIONAL attribute naming a column of the immediate parent <result> element. That is, this applies only for nested queries and serves the purpose of being able to use a query result as input to a nested query.
    - with-param/@param: an OPTIONAL attribute naming a <param> of the parent <query>. That is, this applies only for nested queries and serves the purpose of being able to use a parameter of the parent query input to a nested query.
  - call-query/query: an OPTIONAL <query> being the anonymous query to be invoked as the implementation of this operation with the parameters identified above.

**Defining resources**
This defines the resource identified by "new URI (/data/@baseURI, /data/resource/@path)" and indicates how the request is mapped to a query invocation.

**Defining event trigger**

- event-trigr/@id: REQUIRED id used to identify the event-trigger, used in data services queries.
- event-triger/language REQUIRED currently only XPath is supported as the event trigger language.
- target-topic REQUIRED topic, to which the event notifications will be published.
- subscriptions REQUIRED can be any WS-Eventing compliant endpoint. For example, an SMTP transport can be used to send a message to a mail inbox, where an email address is given as the subscription. Here many subscriptions can be defined for the given topic.

**Security configuration**

When a data service receives messages, it expects to receive a signed and encrypted message as specified by the security policy stored in the registry of your server. Therefore, as shown below, you can embed the security configurations directly in the .dbs file of the data service by adding the path to the relevant security policy. Please see Apache Rampart and Axis2 documentation on the format of the policy file stored in the registry. You can also use the 'enableSec' element to ensure that Apache Rampart is engaged for the data service.

**Sample data service configuration**

Given below is a sample data service configuration with queries, resources etc. for your reference:
<property name="org.wso2.ws.dataservice.driver">org.h2.Driver</property>
<property name="org.wso2.ws.dataservice.protocol">jdbc:h2:file:./samples/database/DATA_SERV_SAMP</property>
<property name="org.wso2.ws.dataservice.user">wso2ds</property>
<property name="org.wso2.ws.dataservice.password">wso2ds</property>
<property name="org.wso2.ws.dataservice.minpoolsize">1</property>
<property name="org.wso2.ws.dataservice.maxpoolsize">10</property>
<property name="org.wso2.ws.dataservice.validation_query"></property>
</config>
=query id="employeesByNumberSQL" useConfig="default">
  <sql>select * from Employees where employeeNumber = ?</sql>
  <result element="employees" rowName="employee">
    <element name="last-name" column="lastName" />
    <element name="first-name" column="firstName" />
    <element name="email" column="email" />
    <element name="salary" column="salary" />
  </result>
  <param name="employeeNumber" paramType="SCALAR" sqlType="INTEGER" type="IN" ordinal="1" />
  <validateLength minimum="3" maximum="20" />
</param>
</query>
<query id="updateProductQuantityQuery" useConfig="default" input-event-trigger="product_stock_low_trigger">
  <sql>update Products set quantityInStock=? where productCode=?</sql>
  <param name="productCode" paramType="SCALAR" sqlType="STRING" type="IN" ordinal="2" />
  <param name="quantityInStock" paramType="SCALAR" sqlType="DOUBLE" type="IN" ordinal="1" />
</query>
<query id="createProductQuery" useConfig="default">
  <sql>insert into Products (productCode, productName, productLine, quantityInStock, buyPrice) values (?, ?, ?, ?, ?)</sql>
  <param name="productCode" paramType="SCALAR" sqlType="STRING" type="IN" ordinal="1" />
  <param name="productName" paramType="SCALAR" sqlType="STRING" type="IN" ordinal="2" />
  <param name="productLine" paramType="SCALAR" sqlType="STRING" type="IN" ordinal="3" />
  <param name="quantityInStock" paramType="SCALAR" sqlType="INTEGER" type="IN" ordinal="4" />
  <param name="buyPrice" paramType="SCALAR" sqlType="DOUBLE" type="IN" ordinal="5" />
</query>
<operation name="employeesByNumber">
  <call-query href="employeesByNumberSQL" >
    <with-param name="employeeNumber" query-param="employeeNumber" />
  </call-query>
</operation>
<event-trigger id="product_stock_low_trigger" language="XPath"/>
<expression>/updateProductQuantityQuery/quantityInStock&lt;10</expression>
<target-topic>product_stock_low_topic</target-topic>
<subscriptions>
  <subscription>mailto:test@test.com</subscription>
</subscriptions>
</event-trigger>
(resource
path="product/{productCode}/{productName}/{productLine}/" method="POST">
  <call-query href="createProductQuery"/>
    <with-param name="productCode" query-param="productCode" />
    <with-param name="productName" query-param="productName" />
    <with-param name="productLine" query-param="productLine" />
    <with-param name="quantityInStock" query-param="quantityInStock" />
    <with-param name="buyPrice" query-param="buyPrice" />
  </call-query>
</resource>
<policy
key="conf:repository/components/org.wso2.carbon.security.mgt/policy/scenario1"/>
Architecture

The WSO2 Data Services Server offers an extremely simple and user-friendly mechanism to take data and make them available as a set of WS-* style Web services or as a set of REST style Web resources. It augments SOA development efforts by providing an easy-to-use platform for creating and hosting data services. This enables easy integration of data into business processes, mashups, gadgets, BI applications and any service in general.

The diagram below depicts an overview of the WSO2 Data Services Server architecture.

Figure: Overview of WSO2 Data Services Server architecture

The WSO2 Data Services Server is 100% open source and developed on top of WSO2 Carbon, the component-based middleware platform consisting of a set of pluggable components. As a result, the Data Services Server inherits identity, statistics, governance, security, clustering and management capabilities of WSO2 Carbon. Because of the Data Services Server’s component-based architecture, developers have the choice to deploy only the required components by adding in and removing features.

Apache Axis2 is the heart of the WSO2 Data Services Server’s SOAP processing engine. In the WSO2 Data Services Server, a data service can be summarized into an XML descriptor file written in compliance with Data Services Descriptor Language (DSDL). DSDL is an XML based language defined by WSO2 to write data services.

A custom deployer extending the Apache Axis2 deployer framework is responsible for reading this data service descriptor and creating a data service. High performant XML processing capabilities offered by Apache AXIOM is used for generating XML responses on the fly. Some of the third-party open source projects used by the WSO2 Data Services Server include Apache DBCP for managing connection pools for Relational Databases, Google Spreadsheet Data API for reading Google spreadsheets, OpenCSV for CSV file support and Apache POI for MS-Excel support.

The Data Services Server’s management console makes comprehensive management, configuration and
monitoring possible though a simple, user-friendly GUI. The GUI is done on a layered architecture by separating the backend and frontend concerns. This allows the user to connect to multiple backends using a single GUI console. The Data Services Server also comes with different types of tools for running, testing and monitoring data services.

**About this Release**

*What is new in this Release*

The WSO2 Data Services Server version 3.5.1 is the successor of version 3.5.0. This release includes the following improvements and bug fixes.

*Issues fixed in this release*

- WSO2 DSS 3.5.1 - Fixed Issues

*Known issues in this release*

- WSO2 DSS 3.5.1 - Known Issues

**Compatible WSO2 product versions**

WSO2 DSS 3.5.1 is based on WSO2 Carbon 4.4.9 and is expected to be compatible with any other WSO2 product that is based on any Carbon 4.4.x version. If you get any compatibility issues, please contact team WSO2. For information on the third-party software required with DSS 3.5.1, see Installation Prerequisites.

**FAQ**

- **DSS deployment**
  - When I set up multiple WSO2 products in my environment (e.g., DSS with WSO2 ESB), should I use dedicated servers?

- **Server administration and troubleshooting**
  - How can I get a SOAP response when invoking an IN-ONLY operation?
  - Why does the server slow down gradually when invoking a service in transport scope?

**DSS deployment**

When I set up multiple WSO2 products in my environment (e.g., DSS with WSO2 ESB), should I use dedicated servers?

It is recommended to set up each product in a separate server. For example, WSO2 ESB is used for message mediation. A considerable amount of processing happens in the ESB. The DSS does data service hosting and has a different architecture layer compared to the ESB. If you deploy both the ESB and DSS in the same instance/runtime, it can negatively impact the performance of both. It also makes scaling difficult.

However, you can set up hybrid servers (i.e., installing selected DSS features on top of the ESB and vice versa) using WSO2 products without the above performance concerns. See deploying and clustering WSO2 products for more information.

**Server administration and troubleshooting**

How can I get a SOAP response when invoking an IN-ONLY operation?

Setting "returnRequestStatus" attribute to true in dataservice operation as shown below will return the response for IN-ONLY operations.
Why does the server slow down gradually when invoking a service in transport scope?

When we have transport session scoped services (when boxcarring is enabled), if the HTTP session is not maintained, Axis2 server creates new context objects in the back-end for each request. These objects pile up and will ultimately run out of memory. Therefore, when boxcarring is enabled, the user always has to retain the session (send back the session cookie) to prevent the server slowing down and eventually running out of memory. If Axis2 client is used to send requests, set 'Options.setManagedSession(true)'.

```xml
<operation name="operation1" returnrequeststatus="true">
  <call-query href="query1">
    <with-param name="dm1" query-param="dm1"/>
    <with-param name="dm2" query-param="dm3"/>
  </call-query>
</operation>
```
Quick Start Guide

The purpose of this guide is to get you started on creating and invoking a data service using WSO2 Data Services Server (WSO2 DSS) as quickly as possible. See the following topics for details:

- Introduction and key concepts
- Creating a sample database with data
- Installing and setting up WSO2 DSS
- Creating your first data service
  - Step 1: Creating the data service
  - Step 2: Adding new datasources
  - Step 3: Creating a query for your datasource
  - Step 4: Creating an operation to invoke a query
- Invoking the data service
  - Using the TryIt tool
  - Sending an HTTP GET request

Introduction and key concepts

WSO2 Data Services Server provides a convenient web service interface for data stored in various datasources. That is, data that is stored in various datasources can be decoupled from the infrastructure where it is stored and exposed to external clients as a data service. Datasources such as relational databases, CSV files, Microsoft Excel files and google spread sheets can be easily service enabled using WSO2 DSS.

Creating a sample database with data

Let us now create a sample MySQL database and add some data. For the purpose of this demonstration, we will create a simple database with one table.

1. Download and install MySQL.
2. Open a terminal and execute the following command to start your MySQL database:

   ```bash
   mysql.server start
   ```
3. Access the MySQL prompt by giving the user name and password. By default the user name is 'root' and the password is blank.

   ```bash
   mysql -u root -p
   ```
4. Create the sample database named `employeedb` using the following command:

   ```sql
   create database employeedb;
   ```
5. Open the `employeedb` database you just created using the following command:
use employeedb;

6. Create one table in the database using the following command:

```sql
CREATE TABLE employee (
    id VARCHAR(10) NOT NULL PRIMARY KEY,
    name VARCHAR(100),
    address VARCHAR(100)
);
```

This command will create the following table in the database:

```
Field | Type    | Null | Key | Default | Extra
------|---------|------|-----|---------|-------
id    | varchar(10) | NO | PRI | NULL |       
name   | varchar(100) | YES |    |       |       
address| varchar(100) | YES |    | NULL |       
```

7. Insert employee data into the `employee` table as follows:

1. The first employee:

   ```sql
   INSERT INTO employee VALUES('01','john','Boston');
   ```

2. The second employee:

   ```sql
   INSERT INTO employee VALUES('02','Micheal','Dallas');
   ```

3. The third employee:

   ```sql
   INSERT INTO employee VALUES('03','richard','Chicago');
   ```

8. You can see the data that you have added to the `employee` table of the `employeedb` database by executing the following command:

```sql
SELECT * FROM employee
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>john</td>
<td>Boston</td>
</tr>
<tr>
<td>02</td>
<td>Micheal</td>
<td>Dallas</td>
</tr>
<tr>
<td>03</td>
<td>richard</td>
<td>Chicago</td>
</tr>
</tbody>
</table>
```

You now have a MySQL database with employee data.
Installing and setting up WSO2 DSS

Follow the steps given below to download WSO2 DSS.

1. Download WSO2 DSS from [here](#).
2. Extract the ZIP file to a location in your computer. This location will be referred to as `<DSS_HOME>` from hereon.
3. Download the JDBC driver for MySQL from [here](#) and copy it to your `<DSS_HOME>/repository/components/lib` directory.

Before you start the server, the following prerequisites should be in place:

1. Ensure that you have JDK 7/8 installed in your computer.
2. You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer.

WSO2 DSS is now installed with the required settings.

Creating your first data service

We are now ready to start creating the first data service. We will be using the `employeedb` database that we created previously and we will expose the data through our data service. Follow the steps given below.

1. Open a terminal and navigate to the `<DSS_HOME>/bin` directory and execute the DSS startup script using one of the following commands:
   
   - On Windows: `wso2server.bat`
   - On Linux: `sh wso2server.sh`

2. When the product is started, the URL of the Management Console will be shown in the terminal as follows:

   ```
   INFO {org.wso2.carbon.ui.internal.CarbonUIServiceComponent} - Mgt Console URL: https://10.100.5.65:9443/carbon/
   ```

3. Copy this URL to your browser to open the Management Console.

4. Log in to the management console using the default administrator credentials: admin/admin.

Step 1: Creating the data service

1. In the left navigator, go to the Main tab and click Services -> Add -> Data Service -> Create. This will open the Create Data Service wizard.

2. In the Data Service Name field, enter 'MyFirstDS' as the name. Leave the default settings of all other options.
3. Click **Next**. This will take you to the page for adding datasources.

**Step 2: Adding new datasources**

1. Click **Add New Datasource**. You can now specify the datasource details.
   1. In the **Datasource ID** field, enter ‘MyDS’.
   2. In the **Datasource Type** field, select **RDBMS** from the list of values.
2. You will now be prompted to enter details of your RDBMS:
3. Enter the following details of your MySQL database in the relevant fields as shown above:
   - **Database Engine**: MySQL
   - **Driver Class**: com.mysql.jdbc.Driver
   - **URL**: jdbc:mysql://localhost:3306/employeedb
   - **User Name**: root
   - **Password**: (Leave this blank)

4. Click **Save** to save the datasource.

5. Click **Next**. This will take you to a new page for adding queries to your service.

**Step 3: Creating a query for your datasource**

1. Now we will create a query to obtain all the employee details saved in the database.
   1. Enter a name for the query in the **Query ID** field. We will use ‘QueryAll’.
   2. All datasources created for the data service will be listed for the **Datasource** field. Select **MyDS** from this list.
   3. Specify the SQL statement in the **SQL** field. We will use a SELECT statement to query for all the employee details in the database as shown below.

```
SELECT id, name, address FROM employee
```

4. We must now specify output mappings to determine how the information in the database should be displayed in the query output. We will be needing three columns (**ID**, **Name** and **Address**) to display
the employee details. Therefore, we need to create three output mapping entries for these fields. Click Generate Response to automatically create output mappings as shown below.

2. Click Next. This will take you to a new page for connecting operations to each of your queries.

Step 4: Creating an operation to invoke a query

1. Enter a name for the operation in the Operation Name field. We will use 'getEmployees'.
2. All the queries defined for the data service will be listed for the Query ID field. Select 'QueryAll'.

3. Save the operation.
4. Click Finish to complete the data service creation. The 'MyFirstDS' data service is now listed as a deployed data service.
Invoking the data service

In this section, we will look at two different ways of invoking our data service without writing a single line of code. First we will invoke our data service using the Tryit utility, which is integrated with the Management Console of WSO2 DSS. We will then invoke the service using a simple HTTP GET request.

Using the TryIt tool

1. In the Deployed Services page, you will now see ‘MyFirstDS’ listed as a data service. You can access the Deployed Services page by clicking Services -> List in the Main tab.
2. Click the Try this service link for the data service. The service will be opened from the TryIt tool as shown below.

All the operations defined for the data service will be listed in the left navigator. As shown above, the ‘MyFirstDS’ data service has one operation (getEmployees).
3. Click getEmployees to invoke the operation.
4. Click Send to execute the getEmployees operation and the result will be printed as shown below.

Note that the employee ID and name will be returned as XML elements and the address will be wrapped as an attribute. This is due to the output mapping that was defined for each column: We have configured ID and Name as element type mappings and the address as an attribute type mapping. You can edit these mappings and observe the associated changes in the response very easily using the TryIt tool.
Sending an HTTP GET request

Now we will invoke the same data service using the HTTP GET request. Get the URL of the data service from the service dashboard:

1. Go to Services -> List to open the Deployed Services page.
2. Click the data service (MyFirstDS) to open the service dashboard:

   ![Service Dashboard](image)

   See that the URL of the 'MyFirstDS' data service is listed under Endpoints: https://10.100.5.65:9443/services/MyFirstDS

3. You can execute the getEmployees operation using the above URL. Simply copy this URL followed by the operation name as shown below.

   https://10.100.5.65:9443/services/MyFirstDS/getEmployees

4. The result will be the same as with the TryIt tool:

   This XML file does not appear to have any style information associated with it. The document tree is shown below.

   ```xml
   <Employee xmlns="http://ws.wso2.org/dataservice">
      <Employee>
         <ID>01</ID>
         <Name>john</Name>
         <Address>Boston</Address>
      </Employee>
      <Employee>
         <ID>02</ID>
         <Name>Micheal</Name>
         <Address>Dallas</Address>
      </Employee>
      <Employee>
         <ID>03</ID>
         <Name>richard</Name>
         <Address>Chicago</Address>
      </Employee>
   </Employee>
   ```
This concludes the quick start guide for WSO2 DSS 3.5.1. You can find more advanced use cases in the Tutorials.
Tutorials

The following tutorials will provide information on the main use cases of WSO2 Data Services Server (WSO2 DSS), which includes creating data services to expose various datasources as services:

- Creating a Data Service from Scratch
- Uploading a Created Data Service
- Generating a Data Service
- Exposing Data as a REST Resource
- Invoking Batch Operations
- Invoking an Operation with Multiple Records
- Defining Nested Queries
- Receiving Notifications from Data Services
- Handling Distributed Transactions
- Exposing Data as an OData Service
- Scheduling Tasks

For a demonstration of other features that you can use with WSO2 DSS, try out the samples.

Creating a Data Service from Scratch

The following tutorials explain how you can expose various datasources as data services using the Create New Data Service wizard of WSO2 DSS:

- Exposing an RDBMS as a Data Service
- Exposing a Google Spreadsheet as a Data Service
- Exposing CSV Data as a Data Service
- Exposing Excel Data as a Data Service
- Exposing RDF Data as a Data Service
- Exposing MongoDB as a Data Service
- Exposing a Web Resource as a Data Service
- Exposing a Carbon Datasource as a Data Service
- Exposing a Custom Datasource as a Data Service
- Exposing Cassandra as a Data Service
- Exposing a JNDI Datasource as a Data Service

Exposing an RDBMS as a Data Service

In this tutorial, we will run through the process of service enabling an RDBMS as a data service.

- Step 1: Setting up an RDBMS
- Step 2: Adding an RDBMS datasource
- Step 3: Defining a query
- Step 4: Defining an operation to invoke the query
- Step 5: Finish creating the data service
- Step 6: Invoking your data service

---

**Step 1: Setting up an RDBMS**

Follow the steps given below to set up a MySQL database for this tutorial.

1. Install the MySQL server.
2. Download the JDBC driver for MySQL from [here](#) and copy it to your `<DSS_HOME>/repository/componen...`
3. Create the following database: trainingdb
4. Create the following table:

CREATE TABLE ACCOUNT(AccountID int NOT NULL, Branch varchar(255) NOT NULL, AccountNumber varchar(255), AccountType ENUM('CURRENT', 'SAVINGS') NOT NULL, Balance FLOAT, ModifiedDate DATE, PRIMARY KEY (AccountID));

5. Enter the following data into the table:

INSERT INTO ACCOUNT VALUES (1, "AOB", "A00012", "CURRENT", 231221, '2014-12-02');

---

**Step 2: Adding an RDBMS datasource**

Follow the steps given below.

1. Log in to the management console and select Create under Data Service.

2. In the Create Data Service screen, enter RDBMS as the data service name.

3. Click Next. When you get to the Add New Data Source screen, select RDBMS as the data source type. The RDBMS-specific options will be available for editing.

   ![Datasource Configuration](image)

   In the **Datasource ID** field, enter AccountDetails as the datasource name.

4. In the **Datasource Type** field, select RDBMS and Default from the lists.

If you enter External instead of the Default datasource type, your datasource should be supported by an external provider class, such as `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource`. You can select the External option and enter the name and value of connection properties by clicking Add Property. For example,
6. Enter MySQL in the Database Engine field.
7. Enter the following driver class for MySQL in the Driver Class URL field: com.mysql.jdbc.Driver.

Be sure to download the JDBC driver for MySQL from [here](#) and copy it to your `<DSS_HOME>/repository/components/lib` directory.

If the driver class does not exist in the relevant folders when you create the datasource, you will get an exception such as ‘Cannot load JDBC driver class com.mysql.jdbc.Driver’.

See the section on working with RDBMS datasources, for more options that you can use with your RDBMS data service.

8. Enter the URL to your MySQL database in the URL field: `jdbc:mysql://localhost:3306/trainingdb`

9. Enter the username and password to connect to your MySQL database. By default the username is root and the password is blank.

10. Click Next to start defining a query.

### Step 3: Defining a query

Now let's start writing a query for getting data from the datasource. The query will specify the data that should be fetched and the format that should be used to display data when the query is invoked.

1. Click Add New Query to open the Add New Query screen.
2. Enter the following values:
   - **Query ID**: Enter an ID for the query.
   - **Datasource**: Select the datasource for which you are going to write a query. Select the RDBMS datasource that you created previously.
   - **SQL**: In this field, enter the SQL statement describing the data that should be retrieved from the RDBMS datasource.
3. **Add input mappings**: Click **Generate Input Mapping** to automatically create a mapping for the **AccountID** field in the database.

<table>
<thead>
<tr>
<th>Mapping Name</th>
<th>Parameter Type</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountID</td>
<td>SCALAR</td>
<td>STRING</td>
<td></td>
</tr>
</tbody>
</table>

Add a new input mapping.

Input mappings allow you to add parameters to a query so that you can set the parameter value when executing the query. According to the above definition, you need to provide the account ID as an input in order to retrieve the data corresponding to the account ID.

Find out more about defining **Input Mappings** in WSO2 DSS.

4. **Add output mappings**: Click **Generate Response** to automatically creating mappings for the fields that should show in the output.

Output mapping is used to specify how the data that is fetched from your query will be shown in the response. Note that, by default, the output type is **XML**. Find out more about defining **Output Mappings** in WSO2 DSS.
5. Click **Next** to open the **Operations** screen.

---

**Step 4: Defining an operation to invoke the query**

Data service operations are written to invoke queries.

1. Click **Add New Operation** to open **Add New Operation** screen.
2. Add a name for your operation in the **Operation Name** field.
3. In the **Query ID** field, select the query that you defined previously.
4. Save the operation.

---

**Step 5: Finish creating the data service**

Once you have defined the operation, click **Finish** to complete the data service creation process. You will now be taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

---

**Step 6: Invoking your data service**

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click the **Try this service** link for the **RDBMS** data service. The TryIt Tool will open with the data service.
3. Select the operation you created earlier. You need to provide the account ID as an input. Enter '1'.
4. Click **Send** to see the details of the account.

```xml
<Entries xmlns="http://ws.wso2.org/dataservice">
  <Entry>
    <AccountID>1</AccountID>
    <Branch>AOB</Branch>
    <AccountNumber>A00012</AccountNumber>
    <AccountType>CURRENT</AccountType>
    <Balance>231221</Balance>
    <ModifiedDate>2014-12-02T05:30</ModifiedDate>
  </Entry>
</Entries>
```

---

**Exposing a Google Spreadsheet as a Data Service**

This tutorial will guide you on how to expose data in a google spreadsheet as a data service in WSO2 Data Services Server.

See the following topics for instructions:

- Start the Create New Data Service wizard
- Add a google spreadsheet datasource
- Define a query for the datasource
  - If query mode is disabled
  - If query mode is enabled
- Define an operation to invoke the query
- Finish creating the data service
- Invoking your data service
Exposing a public spreadsheet as a data service

Start the Create New Data Service wizard

Now, let's start creating the data service from scratch:

1. Log in to the management console using the following URL on your browser: "https://localhost:9443/carbon/".

For the purpose of this tutorial, be sure to use 'localhost' as the IP in the above URL.

2. Click Create under the Data Service menu to open the Create Data Service wizard.

3. Enter GSpread as the data service name as shown below. Leave the default values for the other fields.

4. Click Next to go to the Datasources screen.

Add a google spreadsheet datasource

You can add a google spreadsheet as the datasource by following the steps given below.

1. Click Add New Datasource to open the following screen.

2. Follow the instructions below to fill the datasource details.
1. In the **Datasource Id** field, enter **GoogleSpreadsheet** as the value.

2. In the **Datasource Type** field, specify the type of datasource for which the data service is created. Select **Google Spreadsheet** from the list. You will now get the following screen:

   ![Datasource Configuration Screen](image)

3. In the **Google Spreadsheet URL** field, specify the path to your spreadsheet. The file we are using for this tutorial is https://spreadsheets.google.com/ccc?key=0Av5bU8aVtFjPdErUVN3VmZlRkoyM1ZzVIE1MzdtbXc&hl=en.

   Note that this is a private spreadsheet, which is not published on the web. If you want to use a public spreadsheet, see the topic on exposing a public spreadsheet as a data service.

4. Select a value for the **Visibility** field based on whether the spreadsheet is private or public. Since we have used a private spreadsheet in this example, set the visibility to **Private**. You are asked to provide credentials as well as a browser redirect URL that is known to your browser. Google no longer supports authentication through username and password. Therefore, it is now necessary to provide credentials in the form of a **Client ID**, **Client Secret** and **Refresh Token** as shown below.

   ![Credentials Configuration Screen](image)

5. See the [google documentation](https://developers.google.com/identity/protocols/oauth2) for instructions on how to get an OAuth2 client ID and client secret. In order to get these credentials, you will be asked to provide an **authorization redirect URL**. Be sure to use the same URL as the **browser redirect URL** shown above.

### Setting the hostname

Note that the **Redirect URI** should contain the same host name as the **Authorized Redirect URI** that you provided in the previous step, as well as the host on which the management console runs.

- If the server is running on your machine, you can simply use “localhost” as the hostname (or the direct IP address, which is 127.0.0.1).
6. Update the **Client ID** and the **Client Secret** fields in the **Edit Datasource** screen with the values you got in the previous step:

<table>
<thead>
<tr>
<th>Visibility*</th>
<th>Private (\downarrow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client ID*</td>
<td>255838984457-0p0loig8vev7s9mfcs0km6b51gp:</td>
</tr>
<tr>
<td>Client Secret*</td>
<td>27flTjPzgL03lC65ePZSHYK_</td>
</tr>
<tr>
<td>Refresh Token*</td>
<td></td>
</tr>
<tr>
<td>Redirect URIs*</td>
<td><a href="https://localhost:9443/authCodeReceiver">https://localhost:9443/authCodeReceiver</a></td>
</tr>
</tbody>
</table>

7. Click **Generate Token**. You will now be redirected to the google consent page. After you approve that, the refresh token will be inserted into the **New Datasource** screen automatically as shown below. Note that we just store the refresh token because the access token is going to expire anyway.

<table>
<thead>
<tr>
<th>Visibility*</th>
<th>Private (\downarrow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client ID*</td>
<td>255838984457-0p0loig8vev7s9mfcs0km6b51gp:</td>
</tr>
<tr>
<td>Client Secret*</td>
<td>27flTjPzgL03lC65ePZSHYK_</td>
</tr>
<tr>
<td>Refresh Token*</td>
<td>1%252Fa7kk--2FNO41srP--_tj0zMDxIWbeC4fejCCLI</td>
</tr>
<tr>
<td>Redirect URIs*</td>
<td><a href="https://localhost:9443/authCodeReceiver">https://localhost:9443/authCodeReceiver</a></td>
</tr>
</tbody>
</table>

8. The **Use Query Mode** option allows you to write queries for the spreadsheet in two different modes:

- **Non-Query** mode: Allows you to directly expose the contents of a Google spreadsheet as a service.

- **Query** mode: Allows you to query a Google spreadsheet in a familiar, SQL-like manner, and expose the result as a service. You need to provide the name of the spreadsheet, in addition to the **spreadsheet URL** as shown below:
Define a query for the datasource

You can define queries in two ways for a google spreadsheet, depending on whether or not Query mode is enabled for the datasource.

If query mode is disabled

If query mode is disabled for the spreadsheet, you cannot use SQL statements to query data in the spreadsheet. Note that in non-query mode, you can only get data from the spreadsheet and you cannot insert, update or modify any data.

1. Click Add New Query to open the Add New Query screen.
2. Enter Q1 as the query id in the Query ID field.
3. In the Datasource field, select the datasource for which you are going to write a query. Select the datasource for the Google spreadsheet that you created previously.
4. You can directly specify the details of the spreadsheet as shown below.

See the next section on defining queries for more information on how the query mode affects how you query data in the spreadsheet.

3. Save the datasource.
4. Click Next to go to the Queries screen
5. Define **Output Mapping**: Now, let's specify how the data fetched from the datasource should be displayed in the output. The google spreadsheet we are using contains several columns with customer data. We will create output mappings for the following columns: **ID**, **CustomerNumber**, **CustomerName** and **City**.

   1. In the **Output type** field, specify the format in which the query results should be presented. You can select XML, JSON or RDF. We will use **XML** for this tutorial.
   2. In the **Grouped by element** field, specify a grouping for all the output mappings. This will be the XML element that will group the query result. Enter **Customers** in this field.
   3. In the **Row Name** field, specify the XML element that should group each individual result. Enter **Customer** in this field.
   4. Click **Add New Output Mapping** to start creating the output mapping for the **ID** column. Enter values as shown below:

![Add Output Mapping (GSpread/Q1)](image)

5. Click **Add** to save the output mapping. You will now have one output mapping listed for the query.
6. Now, add output mappings for the following:
5. Define **Input Mapping**: You need to create input mappings for the `customerNumber`, `customerName` and `city` columns specified in the above SQL statement. These input mapping parameters will be used for inserting data into the relevant columns. Click **Generate Input Mapping**, and default input mappings will be generated for all the fields that you have specified in your SQL statement as shown below.

<table>
<thead>
<tr>
<th>Mapping Type</th>
<th>Element Name</th>
<th>Datasource Type</th>
<th>Datasource Column Name</th>
<th>Parameter Type</th>
<th>Schema Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>element</td>
<td><code>customerNumber</code></td>
<td>column</td>
<td><code>customerNumber</code></td>
<td>SCALAR</td>
<td>string</td>
</tr>
<tr>
<td>attribute</td>
<td><code>customerName</code></td>
<td>column</td>
<td><code>customerName</code></td>
<td>SCALAR</td>
<td>string</td>
</tr>
<tr>
<td>attribute</td>
<td><code>city</code></td>
<td>column</td>
<td><code>city</code></td>
<td>SCALAR</td>
<td>string</td>
</tr>
</tbody>
</table>

You will now have the following output mappings listed for the query:

![Output Mappings](image)

Find out more about defining **Output Mappings** in WSO2 DSS.

6. Click **Next** to go to the **Operations** screen.

**If query mode is enabled**

You can query data in the spreadsheet using SQL statements. At the moment, the query mode supports only the basic SELECT, INSERT, UPDATE and DELETE queries. Note that the Google spreadsheet sql driver does not accept the `SELECT *` command. Therefore, all the required select options (e.g., column names) should be specified in the query. See the following example: `<sql>SELECT employeeId,name,salary FROM Sheet1 WHERE employeeId = ?</sql>`. Nested queries will be supported in an upcoming release.

The **org.wso2.carbon.dataservices.sql.driver.TDriver** class is used internally as the SQL driver to implement the query mode. It is a JDBC driver implementation to be used with tabular data models such as Google spreadsheets, Excel sheets etc.

1. Click **Add New Query** to open the **Add New Query** screen.
2. Enter `Q1` as the query id in the **Query ID** field.
3. In the **Datasource** field, select the datasource for which you are going to write a query. Select the datasource for the Google spreadsheet that you created previously.
4. Specify an SQL query to insert data into the spreadsheet.

```
INSERT INTO customers (`customerNumber`, `customerName`, `city`) VALUES(?,?,?)
```

5. Define **Input Mapping**: You need to create input mappings for the `customerNumber`, `customerName` and `city` columns specified in the above SQL statement. These input mapping parameters will be used for inserting data into the relevant columns. Click **Generate Input Mapping**, and default input mappings will be generated for all the fields that you have specified in your SQL statement as shown below.
Find out more about defining Input Mappings in WSO2 DSS.

6. Click **Next** to go to the **Operations** screen.

**Define an operation to invoke the query**

Follow the steps given below.

1. Click **Add New Operation** to open the **Add New Operation** screen.
2. In the **Operation Name** field, enter a name for the operation.
3. In the **Query ID** field, select the query that you created.
4. Save the operation.

**Finish creating the data service**

Once you have defined the operation, click **Finish** to complete the data service creation process. You will now be taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

**Invoking your data service**

You can try the data service you created by using the **TryIt** tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click the **Try this service** link for the **GSpread** data service that you just created. The TryIt tool will open with the **GSpread** service.
3. Select the operation you created earlier and click **Send**.
   - If you are invoking a query that can insert data into the spreadsheet, you will need to specify the values that should be inserted before clicking **Send**.
   - If you are invoking a query that can fetch data from the spreadsheet, the relevant data will be published when you click **Send**.
Exposing CSV Data as a Data Service

This tutorial will guide you on how to expose data in a CSV file as a data service by using the Create New Data Service wizard. We will create a data service that can search for data on the file.

Note that only read operations are supported for CSV. That is, you cannot insert, update or modify data in a CSV file using WSO2 DSS.

We will use the Products.csv file that is shipped with WSO2 DSS by default. The Products.csv file stored in the <DSS_HOME>/samples/resources folder contains data about products (cars/motorcycles) that are manufactured in an automobile company. The data table has the following columns: "ID", "Name", "Classification" and "Price".

Follow the steps given below to create a data service for this datasource.

- Start the Create New Data Service wizard
- Add a CSV file as the datasource
- Define a query for the datasource
- Define an operation to invoke the query
- Finish creating the data service
- Invoking your data service
Start the Create New Data Service wizard

Now, let's start creating the data service from scratch:

1. Log in to the management console of DSS using the following URL on your browser: "https://10.100.5.65:9443/carbon/".

2. Click Create under the Data Service menu to open the Create Data Service window.

3. Enter CSV as the data service name as shown below. Leave the default values for the other fields.

   ![Data Service Window](image)

4. Click Next to go to the Datasources screen.

   ![Datasources Screen](image)

Add a CSV file as the datasource

You can add a CSV file as the datasource as explained below.

1. Click Add New Datasource and enter values as shown below.

   ![Add New Datasource](image)

2. Follow the instructions below to fill the datasource details.
2. **Datasource Id**: Enter an ID for your datasource. We will enter CSV as the datasource Id.
3. **Datasource Type**: Select CSV as the datasource type from the list. You can now specify the details of the CSV file.
4. **CSV File Location**: Specify the path to your CSV file. In this tutorial, we are using a sample CSV file that is stored in the following location of your product distribution: ./samples/resources/Products.csv.
5. **Contains Column Header Row**: Select true from the list.
6. **Header row**: Enter ‘1’ as the value.

3. Save the datasource.
4. Click Next to go to the Queries screen.

**Define a query for the datasource**

Now let's start writing a query for searching for data in the CSV datasource. The query will specify the data that should be fetched, and the format that should be used to display data when the query is invoked.

1. Click Add New Query to open the Add New Query screen.
2. In the Query ID field, enter an identifier for the query. We will enter Q1.
3. In the Datasource field, select the datasource for which you are going to write a query. Select the CSV datasource that you created previously.
4. Now let's specify output mappings, which will determine how the result from your query will be presented when the query is invoked. The sample CSV datasource we are using contains three columns: ID, Name, Category, and Price. We will create an output mapping for each of these columns.
   1. In the Output type field, specify the format in which the query results should be presented. You can select XML, JSON or RDF. We will use XML for this tutorial.
   2. In the Grouped by element field, specify a grouping for all the output mappings. This will be the XML element that will group the query result. We will enter Products.
   3. In the Row Name field, specify the XML element that should group each individual result. Enter Product in this field.
   4. Click Add New Output Mapping to start creating the output mapping for the ID column. Enter values as shown below:
5. Click Add to save the output mapping. You will now have one output mapping listed for the Q1 query.

6. Create output mappings for the remaining columns given below.

<table>
<thead>
<tr>
<th>Mapping Type</th>
<th>Element Name</th>
<th>Datasource Type</th>
<th>Datasource Column Name</th>
<th>Parameter Type</th>
<th>Schema Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Classification</td>
<td>Column</td>
<td>Classification</td>
<td>SCALAR</td>
<td>string</td>
</tr>
<tr>
<td>Element</td>
<td>Price</td>
<td>Column</td>
<td>Price</td>
<td>SCALAR</td>
<td>string</td>
</tr>
<tr>
<td>Element</td>
<td>Name</td>
<td>Column</td>
<td>Name</td>
<td>SCALAR</td>
<td>string</td>
</tr>
</tbody>
</table>

Find out more about defining Output Mappings in WSO2 DSS.

5. Click Next to go to the Operations screen.

Define an operation to invoke the query

To invoke the query, you need to define an operation.

1. Click Add New Operation to open the following screen.
1. In the **Operation Name** field, enter a name for the operation.
2. In the **Query ID** field, select the query that you created.
3. Save the operation.

**Finish creating the data service**

Once you have defined the operation, click **Finish** to complete the data service creation process. You will now be taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

**Invoking your data service**

You can try the data service you created by using the **TryIt** tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click the **Try this service link** for the **CSV** data service. The **TryIt** tool will open with the CSV service.
3. Select the operation that you created previously and click **Send**.
4. The result of the Q1 query will be published.

**Exposing Excel Data as a Data Service**

This tutorial will guide you on how to expose data in an Excel sheet as a data service by using the **Create New Data Service** wizard. We will create a data service that can search for data on the file and insert data into the file.

To demonstrate this feature, we will use the **Products.xls** file that is shipped with WSO2 DSS by default. The **Products.xls** file is stored in the `<DSS_HOME>/samples/resources/` folder and it contains data about products (cars/motorcycles) that are manufactured in an automobile company. The data table has the following columns: "ID", "Model" and "Classification".

See the following topics for instructions:

- Start the Create New Data Service wizard
- Add an Excel file as the datasource
- Define a query for the datasource
  - If query mode is disabled


- If query mode is enabled
- Defining operations to invoke the query
- Finish creating the data service
- Invoking your data service

Start the Create New Data Service wizard

Now, let’s start creating the data service from scratch:

1. Log in to the management console using the following URL on your browser: “https://10.100.5:9443/carbo/n/”.

2. Click Create under the Data Service menu to open the Create Data Service window.

3. Enter Excel as the data service name as shown below. Leave the default values for the other fields.

   ![Create Data Service](image)

   1. Click **Next** to go to the Datasources screen.

Add an Excel file as the datasource

You can add an Excel file as the datasource as explained below.

1. Click **Add New Datasource** to open the following screen.

   ![Add New Datasource](image)

2. Follow the instructions below to fill the datasource details.

   1. In the **Datasource Id** field, enter Excel as the value.

   2. In the **Datasource Type** field, specify the type of datasource for which the data service is created. Select Excel from the list. You will now get the following screen:

   ![Add New Datasource Type](image)

   3. In the **Excel URL** field, specify the path to your Excel file. In this tutorial, we are using a sample excel file that is stored in the following location of your product pack: ./samples/resources/Products.xls.
4. The **Use Query Mode** option allows you to write queries for the datasource in two different ways.
   - Non-Query mode: Allows you to directly expose the contents of the excel file as a service.
   - Query mode: Allows you to query the Excel file in SQL-like manner, and expose the results as a service.

   See the next section on writing queries for more information on how the query mode affects how you query data in the excel sheet.

3. Save the datasource.
4. Click **Next** to go to the **Queries** screen.

**Define a query for the datasource**

You can define queries in two ways for an excel datasource, depending on whether or not **Query mode is enabled** for the datasource.

**If query mode is disabled**

If query mode is disabled for the spreadsheet, you cannot use SQL statements to query data in the excel sheet. Note that in non-query mode, you can only get data from the sheet and you cannot insert, update or modify any data.

Follow the steps given below.

1. Click **Add New Query** to open the **Add New Query** screen.
2. Enter **Q1** as the query id in the **Query ID** field.
3. In the **Datasource** field, select the datasource for which you are going to write a query. Select the **Excel** datasource that you created previously.

4. You can directly specify the details of the spreadsheet as shown below.

   ![Add New Query Screen](image)

5. Define **Output Mapping**: Now, let’s specify how the data fetched from the datasource should be displayed in the output. The sample Excel datasource we are using contains three columns: ID, Model and Classification. We will create an output mapping for each of these columns.
   1. In the **Output type** field, specify the format in which the query results should be presented. You can select XML, JSON or RDF. We will use **XML** for this tutorial.
   2. In the **Grouped by element** field, specify a grouping for all the output mappings. This will be the XML element that will group the query result. Enter **Products** in this field.
   3. In the **Row Name** field, specify the XML element that should group each individual result. Enter **Prod**
uct in this field.
4. Click **Add New Output Mapping** to start creating the output mapping for the ID column. Enter values as shown below:

![Add Output Mapping (Excel/Q2)](image)

5. Click **Add** to save the output mapping.
6. Create output mappings for the remaining columns given below.

<table>
<thead>
<tr>
<th>Mapping Type</th>
<th>Element Name</th>
<th>Datasource Type</th>
<th>Datasource Column Name</th>
<th>Parameter Type</th>
<th>Schema Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Model</td>
<td>Column</td>
<td>Model</td>
<td>SCALAR</td>
<td>string</td>
</tr>
<tr>
<td>Element</td>
<td>Classification</td>
<td>Column</td>
<td>Classification</td>
<td>SCALAR</td>
<td>string</td>
</tr>
</tbody>
</table>

Find out more about defining **Output Mappings** in WSO2 DSS.

6. Click **Next** to go to the **Operations** screen.

**If query mode is enabled**

You can query data in the spreadsheet using SQL statements. At the moment, the query mode supports only basic SELECT, INSERT, UPDATE and DELETE queries. Nested queries will be supported in an upcoming release. The org.wso2.carbon.dataservices.sql.driver.TDriver class is used internally as the SQL Driver. It is a
JDBC driver implementation used with tabular data models such as Google spreadsheets, Excel sheets etc.

Follow the steps given below.

1. Click **Add New Query** to open the **Add New Query** screen.
2. Enter **Q1** as the query id in the **Query ID** field.
3. In the **Datasource** field, select the datasource for which you are going to write a query. Select the **Excel** data source that you created previously.

4. Specify an SQL query to insert data into the spreadsheet.

   ```
   INSERT INTO sheet1 (ID, Model, Classification) VALUES (?, ?, ?)
   ```

5. Define **Input Mapping**: You need to create input mappings for the ID, Model and Classification columns specified in the above SQL statement. These input mapping parameters will be used for inserting data into the relevant columns. Click **Generate Input Mapping**, and default input mappings will be generated for all the fields that you have specified in your SQL statement as shown below.

   Find out more about defining **Input Mappings** in WSO2 DSS.

6. Click **Next** to go to the **Operations** screen.

   **Defining operations to invoke the query**

   Follow the steps given below.

   1. Click **Add New Operation** to open the following screen:
2. In the **Operation Name** field, enter a name for your operation.

3. In the **Query ID** field, select the query you created previously.

4. Save the operation.

**Finish creating the data service**

Once you have defined the operation, click **Finish** to complete the data service creation process. You will now be taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

---

**Invoking your data service**

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click the **Try this service** link for the Excel data service. The TryIt Tool will open with the Excel data service.
3. Select the operation you created earlier and click **Send**.
   - If you are invoking a query that can insert data into the excel sheet, you will need to specify the values that should be inserted before clicking **Send**.
   - If you are invoking a query that can fetch data from the excel sheet, the relevant data will be published when you click **Send**.

**Exposing RDF Data as a Data Service**

This tutorial will guide you on how to expose an RDF datasource as a data service using WSO2 Data Services Server. To demonstrate this feature, we will use a sample RDF file that is shipped with WSO2 DSS by default. Given below are the details of this datasource:

The RDF file (Movies.rdf), stored in the `<DSS_HOME>/samples/resources/` folder contains data about some popular movies. Each movie data has the following sub elements: "title", "director", "year", "genre" and "actor".

```xml
tag xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```
<rdf:Description rdf:about="http://www.popular.movies/cd/Roman" Holiday">
  <cd:title>Roman</cd:title>
  <cd:director>William</cd:director>
  <cd:year>1953</cd:year>
  <cd:genre>Romance</cd:genre>
  <cd:actor>Gorella</cd:actor>
</rdf:Description>

<rdf:Description rdf:about="http://www.popular.movies/cd/Bridget" Jones's Diary">
  <cd:title>Bridget</cd:title>
  <cd:director>Cilla</cd:director>
  <cd:year>2001</cd:year>
  <cd:genre>Drama</cd:genre>
  <cd:actor>John</cd:actor>
</rdf:Description>

<rdf:Description rdf:about="http://www.popular.movies/cd/The Proposal">
  <cd:title>The Proposal</cd:title>
  <cd:director>Ann</cd:director>
  <cd:year>2009</cd:year>
  <cd:genre>Romance</cd:genre>
  <cd:actor>Alexis</cd:actor>
</rdf:Description>

<rdf:Description rdf:about="http://www.popular.movies/cd/Austenland">
  <cd:title>Austenland</cd:title>
  <cd:director>Jerusha</cd:director>
  <cd:year>2013</cd:year>
  <cd:genre>Adaptation</cd:genre>
  <cd:actor>Georgia</cd:actor>
</rdf:Description>

<rdf:Description rdf:about="http://www.popular.movies/cd/Beginners">
  <cd:title>Beginners</cd:title>
  <cd:director>Mike</cd:director>
  <cd:year>2011</cd:year>
  <cd:genre>Drama</cd:genre>
  <cd:actor>Seth T Walker</cd:actor>
</rdf:Description>

<rdf:Description rdf:about="http://www.popular.movies/cd/Flipped">
  <cd:title>Flipped</cd:title>
  <cd:director>Rob Reiner</cd:director>
</rdf:Description>

<rdf:Description rdf:about="http://www.popular.movies/cd/Silver Linings Playbook">

<rdf:Description rdf:about="http://www.popular.movies/cd/Begin Again">

<rdf:Description rdf:about="http://www.popular.movies/cd/Adventureland">

<rdf:Description rdf:about="http://www.popular.movies/cd/The Spectacular Now">

<rdf:Description rdf:about="http://www.popular.movies/cd/WALL-E">

<rdf:Description rdf:about="http://www.popular.movies/cd/Love Actually">
See the following topics for instructions:

- Start the Create New Data Service wizard
- Add and RDF file as the datasource
- Define a query for the datasource
- Define an operation to invoke the query
- Finish creating the data service
- Invoking your data service

Start the Create New Data Service wizard
Now, let’s start creating the data service from scratch:

1. Log in to the management console of DSS using the following URL on your browser: "https://10.100.5.65:9443/carbon/".

2. Click **Create** under the **Data Service** menu to open the **Create Data Service** window.

3. Enter **RDF** as the data service name as shown below. Leave the default values for the other fields.

4. Click **Next** to go to the **Datasources** screen.

---

**Add and RDF file as the datasource**

You can add an RDF file as the datasource as explained below.

1. Click **Add New Datasource** and enter values as shown below.

2. Follow the instructions below to fill the datasource details.
   - **Datasource ID**: Enter **RDF** as the ID for the datasource.
   - **Datasource Type**: Select **RDF** as the datasource type from the list. You can then specify the location of your RDF file: ./samples/resources/Movies.rdf
   - **RDF File Location**: Enter the path to the RDF file. In this tutorial we are using a sample RDF file that is shipped with your product. The file location is ./samples/resources/Movies.rdf.

3. Save the datasource.
4. Click **Next** to go to the **Queries** screen.

---

**Define a query for the datasource**
Now let's start writing a query for getting data from the Movies.rdf file. The query will specify the data that should be fetched by this query, and the format that should be used to display data when the query is invoked.

1. Click Add New Query to open the Add New Query screen.
2. Enter the following values:
   - **Query ID**: Enter GetMoviesbyGenre as the query ID.
   - **Datasource**: Select the datasource for which you are going to write a query. Select the RDF datasource that you created previously.
   - **SPARQL**: In this field, enter the sparql query describing the data that should be retrieved from the RDF datasource. We will use the following query:

```
PREFIX cd: <http://www.popular.movies/cd#>

WHERE { 
}
```

Create input mapping

The input mapping section is used to specify the type of input that should be given in order to get the output result.

1. Click Add New Input Mapping to start creating the input mapping.
2. We will create an input mapping for 'genre', which will allow us to query for data based on the genre of the movie. Enter the values as shown below.

3. The new input mapping will be listed as shown below.
Create output mapping

Now let's specify output mappings to specify how the data fetched from the query should be displayed. We will create output mappings for all the types of data in the RDF file: title, director, year, genre and actor. Follow the steps given below.

1. In the Output type field, specify the format in which the query results should be presented. You can select XML, JSON or RDF. We will use XML for this tutorial.
2. In the Grouped by element field, specify a grouping for all the output mappings. This will be the XML element that will group the query result. Enter Movies in this field.
3. In the Row Name field, specify the XML element that should group each individual result. Enter Movie in this field.
4. Click Add New Output Mapping to start creating the output mapping for the title column. Enter values as shown below:
5. Click Add to save the output mapping. You will now have one output mapping listed for the `GetMoviesbyGenre` query as shown below.

6. Now, add the following output mappings:

<table>
<thead>
<tr>
<th>Mapping Type</th>
<th>Element Name</th>
<th>Datasource Type</th>
<th>Datasource Column Name</th>
<th>Parameter Type</th>
<th>Schema Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>director</td>
<td>Column</td>
<td>director</td>
<td>SCALAR</td>
<td>string</td>
</tr>
<tr>
<td>Element</td>
<td>year</td>
<td>Column</td>
<td>year</td>
<td>SCALAR</td>
<td>string</td>
</tr>
<tr>
<td>Element</td>
<td>genre</td>
<td>Column</td>
<td>genre</td>
<td>SCALAR</td>
<td>string</td>
</tr>
<tr>
<td>Element</td>
<td>actor</td>
<td>Column</td>
<td>actor</td>
<td>SCALAR</td>
<td>string</td>
</tr>
</tbody>
</table>

Find out more about defining Output Mappings in WSO2 DSS.
Define an operation to invoke the query

Follow the steps given below.

1. Once you have defined the query, click Next to go to the Operations screen.

2. Click Add New Operation and enter the following values:
   - Operation Name: Enter GetMoviesbyGenre as the operation name.
   - Query ID: Select the GetMoviesbyGenre query in this field.

3. Save the operation.

Finish creating the data service

Once you have defined the operation, click Finish to complete the data service creation process. You will now be taken to the Deployed Services screen, which shows all the data services deployed on the server.

Invoking your data service

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the Deployed Services screen.
2. Click the Try this service link for the Excel data service. The TryIt Tool will open with the RDF service.

3. Enter Drama as the genre: `<xs:Genre xmlns:xs="http://ws.wso2.org/dataservice">Drama</xs:Genre>`

4. Select the GetMoviesByGenre operation and click Send.

5. The following XML response will be published on the TryIt tool:

```xml
Movies xmlns="http://ws.wso2.org/dataservice">
    <Movie>
        <title>Ratatouille</title>
        <director>Jan Pinkava</director>
        <year>2007</year>
        <genre>Drama</genre>
        <actor>Teddy Newton</actor>
    </Movie>
    <Movie>
        <title>Catch Me If You Can</title>
        <director>Steven Spielberg</director>
        <year>2002</year>
        <genre>Drama</genre>
        <actor>Antoine Drolet Dumoulin</actor>
    </Movie>
    <Movie>
        <title>The Spectacular Now</title>
        <director>James Ponsoldt</director>
        <year>2013</year>
        <genre>Drama</genre>
        <actor>Andre Royo</actor>
    </Movie>
    <Movie>
        <title>Adventureland</title>
        <director>Greg Mottola</director>
        <year>2009</year>
        <genre>Drama</genre>
        <actor>Ryan Mcfarland</actor>
    </Movie>
    <Movie>
        <title>Begin Again</title>
        <director>John Carney</director>
        <year>2014</year>
        <genre>Drama</genre>
        <actor>Eric Burton</actor>
    </Movie>
    <Movie>
        <title>Flipped</title>
```
<director>Rob Reiner</director>
<year>2010</year>
<genre>Drama</genre>
<actor>Michele Messmer</actor>
</Movie>

<Movie>
<title>Beginners</title>
<director>Mike Mills</director>
<year>2011</year>
<genre>Drama</genre>
<actor>Seth T Walker</actor>
</Movie>

<Movie>
<title>Bridget Jones's Diary</title>
<director>Cilla Ware</director>
<year>2001</year>
<genre>Drama</genre>
Exposing MongoDB as a Data Service

This tutorial will guide you on how to expose data from a MongoDB datasource as a data service using WSO2 Data Services Server.

Follow the steps given below.

- Install and start MongoDB
- Start the Create New Data Service wizard
- Add MongoDB as a datasource
- Define a query for the datasource
- Define operations to invoke the query
- Finish creating the data service
- Invoking your data service

Install and start MongoDB

A MongoDB server v2.4.x or v2.2.x should be already running in the default port.

Create a collection as below in the command shell.

Start the Create New Data Service wizard

Now, let's start creating the data service from scratch:

1. Log in to the management console of DSS using the following URL on your browser: “https://10.100.5.65:9443/carbon/”.

2. Click Create under the Data Service menu to open the Create Data Service window.

3. Enter MongoDB as the data service name as shown below. Leave the default values for the other fields.

   ![Create Data Service](image)

4. Click Next to go to the Datasources screen.
Add MongoDB as a datasource

When you get to the Add New Data Source screen, click Add New Datasource to open the corresponding screen and enter the following values:

- **Datasource ID:** Enter mongo as the ID for the datasource.
- **Datasource Type:** Select MongoDB as the datasource type from the list.
- You can then specify the connection details to the MongoDB database you set up previously.

The fields available for the MongoDB datasource type are as follows:

- **Servers:** A comma separated list of server hosts and ports where the database is running. E.g.: "localhost" - "125.10.5.3, 125.10.5.4" - "192.168.3.1:27017, 192.168.3.2:27017".
- **Database Name:** The name of the database to which you want to connect.
- **Write Concern:** Select NONE from the list. The write concern value to control the write behavior as well as exception raising on error conditions:

  - **Descriptions of Write Concern options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSYNC_SAFE</td>
<td>Exceptions are raised for network issues, and server errors. The write operation waits for the server to flush the data to disk.</td>
</tr>
<tr>
<td>NONE</td>
<td>No exceptions are raised, even for network issues.</td>
</tr>
</tbody>
</table>
NORMAL Exceptions are raised for network issues, but not server errors.

REPLICAS_SAFE Exceptions are raised for network issues, and server errors; waits for at least 2 servers for the write operation.

SAFE Exceptions are raised for network issues, and server errors; waits on a server for the write operation.

- **Read Preference**: Select PRIMARY from the list. The read preference value which describes how MongoDB clients route read operations to members of a replica set. It has the following options.

  Descriptions of Read Preference options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMARY</td>
<td>Default mode. All operations read from the current replica set primary.</td>
</tr>
<tr>
<td>SECONDARY</td>
<td>All operations read from the secondary members of the replica set.</td>
</tr>
</tbody>
</table>

- **Auto Connect Retry**: Controls whether or not to connect. That is, the system retries to connect automatically.

- **Connection Timeout**: Connection timeout in milliseconds. 0 is default and infinite.

- **Max. Wait**: Max wait time of a blocking thread for a connection.

- **Socket Timeout**: Socket timeout value. 0 is default and infinite.

- **Connections per Host**: If the number of connections allowed per host is exceeded, further connections will be blocked.

- **Threads Allowed to Block For Connection Multiplier**: The value in this field, multiplied by the connections per host, gives the maximum number of threads that may be waiting for a connection to become available from the pool. All further threads will get an exception. For example, if connections per host is 10 and the ‘threads allowed to block for connection multiplier’ is 5, up to 50 threads can wait for a connection.

**Define a query for the datasource**

Now, let's start writing a query for getting data from the MongoDB datasource. The query will specify the data that should be fetched by this query, and the format that should be used to display data when the query is invoked.

We will create two queries:

- **mongo_insert**: This query will be defined to insert data into the database. We will be adding values for the **customer ID** and **customer Name** fields.
- **mongo_find**: This query will search for requested data in the database and display them.

**To create the mongo_insert query:**

1. Click **Add New Query** to open the **Add New Query** screen.
2. We will first create a query that can add data into the database. Enter the following values:
   - **Query ID**: Enter `mongo_insert` as the query ID.
   - **Datasource**: Select the datasource for which you are going to write a query. Select the `mongo` datasource that you created previously.
   - **SPARQL**: In this field, enter the SPARQL query describing the data that should be retrieved from the RDF datasource. We will use the following query:

```
things.insert("{id:#, name:#}")
```
3. Now let's specify input mappings for the data in the MongoDB database. We will create output mappings for the id and name fields.

1. Click **Add New Input Mapping** to start creating the input mapping. We want to add data into the database using the **id** and **name** fields. First, create the input mapping for **id** as shown below.

```
<table>
<thead>
<tr>
<th>Mapping Name*</th>
<th>Parameter Type</th>
<th>SQL Type*</th>
<th>Default Value</th>
<th>IN/OUT Type</th>
<th>Ordinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>SCALAR</td>
<td>STRING</td>
<td></td>
<td>IN</td>
<td></td>
</tr>
</tbody>
</table>
```

2. Click **Add** to save the input mapping.
3. Create the input mapping for the **name** field.
4. The two input mappings will be listed as shown below.

```
<table>
<thead>
<tr>
<th>Mapping Name*</th>
<th>Parameter Type</th>
<th>SQL Type*</th>
<th>Default Value</th>
<th>IN/OUT Type</th>
<th>Ordinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>SCALAR</td>
<td>INTEGER</td>
<td></td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>SCALAR</td>
<td>STRING</td>
<td></td>
<td>IN</td>
<td></td>
</tr>
</tbody>
</table>
```

Find out more about defining **Input Mappings** in WSO2 DSS.

4. Click **Next** to go the **Operations** screen.

**To create the mongo_find query:**

1. Click **Add New Query** to open the **Add New Query** screen.
2. Enter the following values to define a query that can find data from the database:
   - **Query ID**: Enter **mongo_find** as the query ID.
   - **Datasource**: Select the datasource for which you are going to write a query. Select the **mongo** datasource that you created previously.
   - **SPARQL**: In this field, enter the SPARQL query describing the data that should be retrieved from the MongoDB datasource. We will use the following query:
3. Define **Output Mapping**: Now, let's specify how the data fetched from the datasource should be displayed in the output. We will create output mappings for the **Data** field.
   1. In the **Output type** field, specify the format in which the query results should be presented. You can select XML, JSON or RDF. We will use **XML** for this tutorial.
   2. In the **Grouped by element** field, specify a grouping for all the output mappings. This will be the XML element that will group the query result. Enter **Documents** in this field.
   3. In the **Row Name** field, specify the XML element that should group each individual result. Enter **Document** in this field.
   4. Click **Add New Output Mapping** to start creating the output mapping. Enter the following values:

   ![Output Mapping Configuration](image)

   5. Click **Add** to save the mapping.

   Find out more about defining **Output Mappings** in WSO2 DSS.

4. Click **Next** to go to the **Operations** screen.

**Define operations to invoke the query**

Data service operations are written to invoke queries. We will create two operations for this datasource:

- **mongo_insert_op**: This operation will invoke the mongo_insert query.
- **mongo_find_op**: This operation will invoke the mongo_find query.

Follow the steps given below.

1. Click **Add New Operation** to open relevant screen.
2. In the **Operation Name** field, enter mongo_insert_op.
3. In the **Query ID** field, select the mongo_insert query, which you created in the previous step.
4. Save the operation.
5. In the same way, create the mongo_find_op operation to invoke the mongo_find query.
6. You will now have two operations listed.

**Finish creating the data service**

Once you have defined the operation, click **Finish** to complete the data service creation process. You will now be
taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

---

**Invoking your data service**

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click the Try this service link for the MongoDB data service. The TryIt Tool will open with the MongoDB service.
3. Select the `mongo_insert_op` operation and enter values for ID and Name.
4. Click **Send** to execute the operation. The data will be inserted to the database.
5. Now, select the `mongo_find_op` operation and click **Send**. The response will be published in the TryIt tool.

---

**Exposing a Web Resource as a Data Service**

This tutorial will guide you on how to expose a web resource as a data service using WSO2 Data Services Server (WSO2 DSS). You can create a data service that can fetch selected data from a web resource and display the data in XML, JSON or RDF form.

- Start the Create New Data Service wizard
- Add a Web Resource as the datasource
- Define a query for the datasource
- Defining and operation to invoke the query
- Finish creating the data service
- Invoking your data service

---

**Start the Create New Data Service wizard**

Now, let's start creating the data service from scratch:

1. Log in to the management console of DSS using the following URL on your browser: "https://10.100.5.65:9443/carbon/".
2. Click **Create** under the **Data Service** menu to open the **Create Data Service** window.
3. Enter **WebResource** as the data service name as shown below.

![Create Data Service](image)

4. Click **Next** to go to the **Datasources** screen.
Add a Web Resource as the datasource

You can add a web resource as the datasource by following the steps given below.

1. Click **Add New Datasource** to open the following screen:

   ![Add New Datasource Screen]

   - **Datasource ID**: Enter WebHarvestDataSource as the datasource Id.
   - **Datasource Type**: Select Web Datasource from the list.

   You will now get the following screen:

   ![Edit Datasource (WebHarvestDataSource) Screen]

2. Enter the following values:
   1. In the **Datasource ID** field, enter WebHarvestDataSource as the datasource Id.
   2. In the **Datasource Type** field, specifies the type of datasource for which you will create the data service. Select Web Datasource from the list.

3. Select **Inline Web Harvest Config** and enter the configuration as shown below:
<config>
  <var-def name='AppInfo'>
    <xslt>
      <xml>
        <html-to-xml>
          <http method='get' url='https://play.google.com/store/apps'/>
        </html-to-xml>
      </xml>
      <stylesheet>
        <![CDATA[
          <xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
            <xsl:output method="xml" omit-xml-declaration="yes"
              indent="yes"/>
            <xsl:template match="/">
              <AppInfo>
                <xsl:for-each select="*[@class='details']">
                  <App>
                    <Title><xsl:value-of select="a[@class='title']"/></Title>
                    <Description><xsl:value-of select="div[@class='description']"/></Description>
                  </App>
                </xsl:for-each>
              </AppInfo>
            </xsl:template>
        ]]>
      </stylesheet>
    </xslt>
  </var-def>
</config>

3. Save the datasource.
4. Click Next to go to the Queries screen.

Define a query for the datasource

Now let's start writing a query for getting data from the web resource. The query will specify the data that should be fetched by this query, and the format that should be used to display the data.

1. Click Add New Query to open the Add New Query screen.
2. Enter the following values:
   - **Query ID**: In this field, enter `webquery` as the query ID.
   - **Datasource**: In this field, select the datasource for which you are going to write a query. Select the web datasource that you created previously.
   - **Scraper Variable**: When you add a query to a Web datasource, you must enter a **Scraper Variable**. This scraper variable must be the same as the output name in the web datasource configuration, which returns the output from the configuration. In this example, the `var-def` name in the configuration is `weatherInfo(< var-def name = 'AppInfo' >)`.

3. Define **Output Mapping**: Now, let's specify how the data fetched from the datasource should be displayed in the output. We will create output mappings for the following data in the web resource: **Title** and **Description**.
   1. In the **Output type** field, specify the format in which the query results should be presented. You can select XML, JSON or RDF. We will use **XML** for this tutorial.
   2. In the **Grouped by element** field, specify a grouping for all the output mappings. This will be the XML element that will group the query result. Enter **AppInfo** in this field.
   3. In the **Row Name** field, specify the XML element that should group each individual result. Enter **App** in this field.
   4. Click **Add New Output Mapping** to start creating the output mapping for the **Title** field. Enter values as shown below:

   ![Add Output Mapping (WebResource/webQuery)](image)

   - **Mapping Type**: element
   - **Output Field Name**: Title
   - **Element Namespace**:
   - **Datasource Column Name**: Title
   - **Parameter Type**: SCALAR
   - **Schema Type**: string

   5. Click **Add** to save the output mapping.

   6. Now, add another output mapping for the **Description** column.

   7. You will now have the following output mappings listed for the `webquery` query:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Datasource Type</th>
<th>Datasource Column Name</th>
<th>Mapping Type</th>
<th>Allowed User Roles</th>
<th>Schema Type</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>column</td>
<td>Title</td>
<td>element</td>
<td>string</td>
<td></td>
<td><img src="image" alt="Edit" /> <img src="image" alt="Delete" /></td>
</tr>
<tr>
<td>Description</td>
<td>column</td>
<td>Description</td>
<td>element</td>
<td>string</td>
<td></td>
<td><img src="image" alt="Edit" /> <img src="image" alt="Delete" /></td>
</tr>
</tbody>
</table>

   Find out more about defining Output Mappings in WSO2 DSS.

4. Click **Next** to go to the **Operations** screen.
Defining and operation to invoke the query

Follow the steps given below.

1. Click Add New Operation to open the Add New Operation screen.
2. In the Operation Name field, enter GetResource.
3. In the Query ID field, select WebQuery.
4. Save the operation.

Finish creating the data service

Once you have defined the operation, click Finish to complete the data service creation process. You will now be taken to the Deployed Services screen, which shows all the data services deployed on the server.

Invoking your data service

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the Deployed Services screen.
2. Click the Try this service link for the WebResource data service. The TryIt Tool will open with the data service.
3. Select the GetResource operation and click Send.
4. The response will be published.

Exposing a Carbon Datasource as a Data Service

A Carbon datasource is an RDBMS or a custom datasource created using the management console of WSO2 Data Services Server (WSO2 DSS). You can simply use that as the datasource for a data service. A Carbon datasource is persistent, and can be used whenever required.

Creating a Carbon datasource

See Managing Datasources for instructions on how to create a Carbon datasource. Be sure to copy the JDBC driver relevant to the database engine to one of the locations given below.

- If the driver is a JAR file, add it to the <PRODUCT_HOME>/repository/components/lib directory, so that it will be converted to an OSGi bundle and copied to the dropins directory during server startup. For example, if you are using MySQL, you would specify com.mysql.jdbc.Driver as the driver and would copy mysql-connector-java-5.XX-bin.jar to this directory.
- If the driver is already an OSGi bundle, you can directly add it to the <PRODUCT_HOME>/repository/components/dropins directory.

If you do not copy the driver to the relevant directory, you will get an exception similar to “Cannot load JDBC driver class com.mysql.jdbc.Driver” when you create the datasource.

Creating a data service using the Carbon datasource
Follow the instructions in exposing an RDBMS as a data service and when you get to the **Add New Data Source** screen, select **Carbon Data Source** as the data source type. You get a drop-down list from which an existing Carbon data source can be selected as shown below.

Exposing a Custom Datasource as a Data Service

See the topics given below to expose a custom datasource as a data service.

- About custom datasources
- Adding a custom tabular datasource to the data service
- Invoking the custom tabular data service
- Adding a custom query datasource to the data service
- Invoking the custom query data service

**About custom datasources**

Custom datasources allow you to interface data services with your own datasource implementation. There are two options for writing a custom datasource, and these two options cover most of the common business use cases as follows:

- **Custom tabular datasources**: Used to represent data in tables, where a set of named tables contain data rows that can be queried later. A tabular datasource is typically associated with an SQL data services query. This is done by internally using our own SQL parser to execute SQL against the custom datasource. You can use the `org.wso2.carbon.dataservices.core.custom.datasource.TabularDataBasedDS` interface to implement tabular datasources. For a sample implementation of a tabular custom datasource, see `org.wso2.carbon.dataservices.core.custom.datasource.InMemoryDataSource`. Also, this is supported in Carbon datasources with the following datasource reader implementation: `org.wso2.carbon.dataservices.core.custom.datasource.CustomTabularDataSourceReader`.

- **Custom query datasources**: Used when the datasource has some form of query expression support. Custom query datasources are implemented using the `org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryBasedDS` interface. 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 your own. For example, you can support any NoSQL type datasource this way. For a sample implementation of a query-based custom datasource, see `org.wso2.carbon.dataservices.core.custom.datasource.EchoDataSource`. This is supported in Carbon datasources with the following datasource reader implementation: `org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryDataSourceReader`.

In the `init` methods of all custom datasources, user-supplied properties are parsed to initialize the datasource accordingly. Also, a property named `__DATASOURCE_ID__`, which contains a UUID to uniquely identify the current datasource, is passed. Custom datasource authors use this to identify the datasources accordingly. For example, scenarios like datasource instances communicating within a server cluster for data synchronisation.

Find the custom connectors used in WSO2 DSS from the github project located at [https://github.com/wso2/](https://github.com/wso2/)
Adding a custom tabular datasource to the data service

Follow the steps given below to create a custom tabular datasource.

1. Log in to the management console using the following URL on your browser: "https://localhost:9443/carbon/".
2. Click Create under Data Service menu to open the Create Data Service window.
3. Enter CustomTabularDataService as the data service name.
4. When you get to the Add New Datasource screen, enter CustomTabular as the datasource ID.
5. Select Custom Data Source as the datasource type.
6. Select Custom Tabular Data Source.
7. In the Custom Data Source Class field, enter the following interface: org.wso2.carbon.dataservices.core.custom.datasource.InMemoryDataSource
8. Click Add New Property to enter properties that will identify the data in the datasource.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inmemory_datasource_schema</td>
<td>{Users:[ID,Name]}</td>
</tr>
<tr>
<td>inmemory_datasource_records</td>
<td>{Users:[[&quot;1&quot;,&quot;Will Smith&quot;],[&quot;2&quot;,&quot;Denzel Washington&quot;]]}</td>
</tr>
</tbody>
</table>

9. The new datasource will now be listed as shown below.

![Datasources Table]

10. Click Add New Query to start defining a query. We will write a query to find and display all users in the datasource.

1. In the Query ID field, enter getAllUsers.
2. In the Datasource field, select the CustomTabular datasource that you created in the previous step.
3. In the SQL field, enter a query:

   ```sql
   SELECT ID,Name FROM Users
   ```

4. Now let's specify output mappings for the data. We will create output mappings for the following data: ID and Name.
1. In the Output type field, specify the format in which the query results should be presented. You can select XML, JSON or RDF. We will use XML for this tutorial.
2. In the Grouped by element field, specify a grouping for all the output mappings. This will be the XML element that will group the query result. Enter Users in this field.
3. In the Row Name field, specify the XML element that should group each individual result. Enter User in this field.
4. Click Add New Output Mapping to start creating the output mapping for the ID field. Enter...
values as shown below:

5. Click Add to save the output mapping.
6. Now, add another output mapping for the Name column.

7. You will now have the following output mappings listed for the `getAllUsers` query:

   ![](image)

11. Click Add New Operation to open the following screen.

   ![](image)

12. In the Operation Name field, enter ‘getAllUsersOp’.
13. In the Query ID field, select `getAllUsers` from the list.
14. Save the operation.
15. Once you have defined the operation, click Finish to complete the data service creation process. You will now be taken to the Deployed Services screen, which shows all the data services deployed on the server.

**Invoking the custom tabular data service**

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the Deployed Services screen.
2. Click the Try this service link for the CustomTabular data service. The TryIt Tool will open with the data service.
3. Select the getAllUsersOp operation.
4. Click **Send** to see the result:

```xml
<Users xmlns="http://ws.wso2.org/dataservice">
  <User>
    <ID>1</ID>
    <Name>Will Smith</Name>
  </User>
  <User>
    <ID>2</ID>
    <Name>Denzel Washington</Name>
  </User>
</Users>
```

---

### Adding a custom query datasource to the data service

Follow the steps given below to create a custom query datasource.

1. Click **Create** under **Data Service** menu to open the **Create Data Service** window.
2. Enter **CustomQueryDataService** as the data service name.
3. When you get to the **Add New Datasource** screen, enter **CustomQuery** as the datasource ID.
4. Select **Custom Data Source** as the datasource type.
5. Select **Custom Query Data Source**.
6. In the **Custom Data Source Class** field, enter the following interface: `org.wso2.carbon.dataservices.core.custom.datasource.EchoDataSource`.
7. Click **Add New Property** to enter properties that will identify the data in the datasource.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>prop1</td>
<td>prop_value1</td>
</tr>
<tr>
<td>prop2</td>
<td>prop_value2</td>
</tr>
</tbody>
</table>

```xml
<property name="prop1">prop_value1</property>
<property name="prop2">prop_value2</property>
```

8. The new datasource will be listed as shown below.

---

9. Click **Add New Query** to start defining a query. We will write a query to find and display all users in the datasource.
1. In the **Query ID** field, enter `getValues`.
2. In the **Datasource** field, select the **CustomQuery** datasource that you created in the previous step.

3. In the **Expression** field, enter a query:

   ```
column1, column2; R1C1 :param1, R1C2 :param2; R2C1 :param1, R2C2 :param2
   ```

4. Now let's specify input mappings to enter data. We will create input mappings for the fields given in the expression above: **Column1** and **Column2**
   1. Click **Add New Input Mapping** to open the respective screen.
   2. Define a mapping for column 1 as shown below.

   ![Add Input Mapping (CustomQueryDataService/Q1)](image)

   3. Click Add to save the mapping.
   4. Add another input mapping for column 2. You will now have two input mappings for the two columns as shown below.

<table>
<thead>
<tr>
<th>Mapping Name</th>
<th>Parameter Type</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>column1</td>
<td>SCALAR</td>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>column2</td>
<td>SCALAR</td>
<td>STRING</td>
<td></td>
</tr>
</tbody>
</table>

5. Now let's specify **output mappings**, which will determine how the result from your query will be presented when the query is invoked. The sample datasource we are using contains the following columns: **Column 1** and **Column 2**. We will create an output mapping for each of these columns.
   1. In the **Output type** field, specify the format in which the query results should be presented. You can select XML, JSON or RDF. We will use XML for this tutorial.
   2. In the **Grouped by element** field, specify a grouping for all the output mappings. We will enter **Rows**.
   3. In the **Row Name** field, specify the XML element that will be used to display an individual value. We will enter **Row**.
   4. Click **Add New Output Mapping** to start creating the output mapping for column. Enter values as shown below.
5. Click **Add** to save the output mapping.
6. You will now have two output mappings created as shown below.

10. Click **Add New Operation** to open the **Add New Operation** screen.
11. In the **Operation Name** field, enter ‘getValuesOp’.
12. In the **Query ID** field, select getValues from the list.

13. Save the operation.
14. Once you have defined the operation, click **Finish** to complete the data service creation process. You will now be taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

---

**Invoking the custom query data service**

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click the **Try this service** link for the **CustomQueryDataService** data service. The TryIt Tool will open with
the data service.
3. Select the `getValuesOp` operation.
4. Enter any values for the input parameters as shown below.

```
<!--Exactly 1 occurrence-->
<xs:column1 xmlns:xs="http://ws.wso2.org/dataservice">1</xs:column1>
<!--Exactly 1 occurrence-->
```

5. Click **Send** to see the result:

```
<Rows xmlns="http://ws.wso2.org/dataservice">
  <Row>
    <C1>R1C1 :param1</C1>
    <C2>R1C2 :param2</C2>
  </Row>
  <Row>
    <C1>R2C1 :param1</C1>
    <C2>R2C2 :param2</C2>
  </Row>
</Rows>
```

## Exposing Cassandra as a Data Service

This tutorial will guide you on how to expose data stored in Cassandra as a data service.

- Install and start Cassandra
- Adding a Cassandra datasource
- Define a query for the datasource
- Define operation to invoke the query
- Finish creating the data service

### Install and start Cassandra

A Cassandra server of version 1.2.x, 2.0 or 3.0 should be already running in the default port. Cassandra version 3.0 is recommended from DSS 3.5.1 onwards.

### Adding a Cassandra datasource

Now, let’s start creating the data service from scratch:

1. Log in to the management console using the following URL on your browser: “https://localhost:9443/carbon/”.
2. Click **Create** under the **Data Service** menu to open the **Create Data Service** wizard.
3. In the **Create Data Service** screen, enter Cassandra as the data service name.
4. Click **Next** to go to the **Add New Data Source** screen.
5. Select **Cassandra** as the data source type. The Cassandra-specific options will be available for editing.

### Properties for a Cassandra datasource

The following describes the properties supported by the Cassandra datasource.

<table>
<thead>
<tr>
<th>Data Source Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra Servers*</td>
<td>Host names of Cassandra servers in a comma-separated list.</td>
</tr>
<tr>
<td>Keyspace</td>
<td>The default key space used by the Cassandra session.</td>
</tr>
<tr>
<td>Port</td>
<td>The port of Cassandra servers.</td>
</tr>
<tr>
<td>Cluster Name</td>
<td>The Cassandra cluster name.</td>
</tr>
<tr>
<td>Compression</td>
<td>Compression used in communication.</td>
</tr>
<tr>
<td>User Name</td>
<td>The authenticating user's username.</td>
</tr>
<tr>
<td>Password</td>
<td>The authenticating user's password.</td>
</tr>
<tr>
<td>Load Balancing Policy</td>
<td>The client load balancing policy on how calls should be made to the provided servers.</td>
</tr>
<tr>
<td>Enable JMX Reporting</td>
<td>Enable JMX statistics for the connector.</td>
</tr>
<tr>
<td>Enable Metrics</td>
<td>Enable metrics for the connector.</td>
</tr>
<tr>
<td>Local Core Connections Per Host</td>
<td>Connection pooling: Local core connections per server host.</td>
</tr>
<tr>
<td>Remote Core Connections Per Host</td>
<td>Connection pooling: Remote core connections per server host.</td>
</tr>
<tr>
<td>Local Max.Connections Per Host</td>
<td>Connection pooling: Maximum local connections per server host.</td>
</tr>
<tr>
<td>Remote Max.Connections Per Host</td>
<td>Connection pooling: Remote max connections per server host.</td>
</tr>
</tbody>
</table>

See the following links for setting up Cassandra as a data store:

- Get familiar with the basics of Cassandra, including the prerequisites for using Cassandra, by going through the following: [https://wiki.apache.org/cassandra/GettingStarted](https://wiki.apache.org/cassandra/GettingStarted).
- Functionality of the underlying connector used for the Cassandra datasource: [https://www.datastax.com/documentation/developer/java-driver/2.0/index.html](https://www.datastax.com/documentation/developer/java-driver/2.0/index.html).
- Also, note that Cassandra requires the most stable version of Java 7 or 8 you can deploy; preferably the Oracle/Sun JVM.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local New Connection Threshold</td>
<td>This property determines the threshold in the connection pool, which will trigger the creation of a new connection when the connection pool has not reached the <strong>maximum capacity for local hosts</strong>. Generally, it will not be required to change the default value for this parameter.</td>
</tr>
<tr>
<td>Remote New Connection Threshold</td>
<td>This property determines the threshold in the connection pool, which will trigger the creation of a new connection when the connection pool has not reached the <strong>maximum capacity for remote hosts</strong>. Generally, it will not be required to change the default value for this parameter.</td>
</tr>
<tr>
<td>Local Max Requests Per Connection</td>
<td>This property allows you to throttle the number of concurrent requests per connection for local hosts.</td>
</tr>
<tr>
<td>Remote Max Requests Per Connection</td>
<td>This property allows you to throttle the number of concurrent requests per connection for remote hosts.</td>
</tr>
<tr>
<td>Protocol Version</td>
<td>The native protocol version to use. By default, it auto connects. &quot;2&quot; for Cassandra versions 2.0 and upwards, and &quot;1&quot; for Cassandra version 1.2.x.</td>
</tr>
<tr>
<td>Consistency Level</td>
<td>The consistency level used for queries.</td>
</tr>
<tr>
<td>Fetch Size</td>
<td>Fetch size used by queries.</td>
</tr>
<tr>
<td>Serial Consistency Level</td>
<td>The serial consistency level used for queries.</td>
</tr>
<tr>
<td>Reconnection Policy</td>
<td>The reconnection policy used for the cluster.</td>
</tr>
<tr>
<td>Constant Reconnection Policy Delay</td>
<td>If &quot;ConstantReconnectionPolicy&quot; is used for Reconnection Policy, this represents the <strong>constant wait time between reconnection attempts in milliseconds</strong>.</td>
</tr>
<tr>
<td>Exponential Reconnection Policy Base Delay</td>
<td>If &quot;ExponentialReconnectionPolicy&quot; is used for Reconnection Policy, this represents the <strong>base delay in milliseconds</strong>.</td>
</tr>
<tr>
<td>Exponential Reconnection Policy Max. Delay</td>
<td>If &quot;ExponentialReconnectionPolicy&quot; is used for Reconnection Policy, this represents the <strong>maximum delay in milliseconds</strong>.</td>
</tr>
<tr>
<td>Retry Policy</td>
<td>Configured the retry policy in this cluster.</td>
</tr>
<tr>
<td>Connection Timeout</td>
<td>The socket connection timeout in milliseconds.</td>
</tr>
<tr>
<td>Keep Alive</td>
<td>Set if socket keeps alive.</td>
</tr>
<tr>
<td>Read Timeout</td>
<td>Set per host socket read timeout in milliseconds.</td>
</tr>
<tr>
<td>Receive Buffer Size</td>
<td>The socket receive buffer size.</td>
</tr>
<tr>
<td>Send Buffer Size</td>
<td>The socket send buffer size.</td>
</tr>
</tbody>
</table>
**Reuse Address** | The socket re-use address.
---|---
**So Linger** | The socket linger on value.
**TCP no Delay** | Set socket TCP to no delay.
**Enable SSL** | Enable SSL.
**Enable OData** | From DSS 3.5 onwards, OData protocol version 4 (OASIS standards) is supported, which mainly provides support for CRUD operations. You can easily expose databases as an odata service by selecting this check box. Currently, Odata service feature support is available for RDBMS datasources and Cassandra datasources. If you have enabled Odata for your data service, you can complete the data service creation process. That is, you are not required to define queries or operations for the service. This Odata service will now be accessible from the following endpoints:
- For super tenant: `http://localhost:9763/odata/{dataserviceName}/{datasourceId}/`
- For normal tenants: `http://localhost:9763/odata/t/{tenantId}/{dataserviceName}/{datasourceId}/`

**Disable Native Batch Requests** | Disables native Cassandra batch requests, and reverts to emulated batch requests.

* Mandatory fields.

**Define a query for the datasource**

Now let's start writing a query for getting data from the datasource. The query will specify the data that should be fetched by this query, and the format that should be used to display data when the query is invoked.

1. Click **Add New Query** to open the **Add New Query** screen.
2. Enter the following values:
   - **Query ID**: Enter an ID for the query.
   - **Datasource**: Select the datasource for which you are going to write a query. Select the Cassandra datasource that you created previously.
   - **SQL**: In this field, enter the SQL statement describing the data that should be retrieved from the RDBMS datasource.

3. **Add input mappings**: Input mappings allow you to add parameters to a query so that you can set the parameter value when executing the query.

   Find out more about defining **Input Mappings** in WSO2 DSS.

4. **Add output mappings**: Out mapping is used to specify how the data that is fetched from your query will be shown in the response.

   Find out more about defining **Output Mappings** in WSO2 DSS.

5. Click **Next** to open the **Operations** screen.

**Define operation to invoke the query**

Follow the steps given below.
1. Click **Add New Operation** to open the **Add New Operation** screen.
2. Add a name for your operation in the **Operation Name** field.
3. In the **Query ID** field, select the query that you defined previously.
4. Save the operation.

**Finish creating the data service**

Once you have defined the operation, click **Finish** to complete the data service creation process. You will now be taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

**Exposing a JNDI Datasource as a Data Service**

Java Naming and Directory Interface (JNDI) is a Java API (application programming interface) providing naming and directory functionality for Java software clients to discover and look up data and objects via a name. WSO2 Data Services Server supports the JNDI InitialContext implementation by inheriting the JNDI implementation of Tomcat ([http://tomcat.apache.org/tomcat-7.0-doc/jndi-resources-howto.html](http://tomcat.apache.org/tomcat-7.0-doc/jndi-resources-howto.html)). JNDI helps decouple object creation from the object look-up. When you have registered a datasource with JNDI, others can discover it through a JNDI lookup and use it.

- Start the Create New Data Service wizard
- Add a JNDI datasource
- Define a query for the datasource
- Define an operation to invoke the query
- Finish creating the data service

---

**Start the Create New Data Service wizard**

Now, let’s start creating the data service from scratch:

1. Log in to the management console using the following URL on your browser: "https://localhost:9443/carbon/".
2. Click **Create** under the **Data Service** menu to open the **Create Data Service** wizard.
3. Enter a name for the data service. Leave the default values for the other fields.
4. Click **Next** to go to the **Datasources** screen.

**Add a JNDI datasource**

You can add a JNDI datasource by following the steps given below.

1. Click **Add New Datasource** to open the following screen:

   ![Add New Datasource](image)

2. Select **JNDI Datasource** as the data source type. The JNDI-specific options will be available for editing as shown below.
2.

The options in the above window are explained below:

- **JNDI Context Class**: The corresponding context factory class.
- **Provider URL**: The URL that specifies the location of a resource on the Web.
- **Resource Name**: The name of the JNDI resource.

You can also expose an RDBMS Carbon datasource as a JNDI datasource using the **Configure > Datasources** menu of the management console. For instructions, see **Adding datasources**.

**Define a query for the datasource**

Now let's start writing a query for getting data from the datasource. The query will specify the data that should be fetched by this query, and the format that should be used to display data when the query is invoked.

1. Click **Add New Query** to open the **Add New Query** screen.
2. Enter the following values:
   - **Query ID**: Enter an ID for the query.
   - **Datasource**: Select the datasource for which you are going to write a query. Select the **JNDI** datasource that you created previously.
   - **SQL**: In this field, enter the SQL statement describing the data that should be retrieved from the RDBMS datasource.
3. **Add input mappings**: Input mappings allow you to add parameters to a query so that you can set the parameter value when executing the query.
   
   Find out more about defining **Input Mappings** in WSO2 DSS.

4. **Add output mappings**: Out mapping is used to specify how the data that is fetched from your query will be shown in the response.
   
   Find out more about defining **Output Mappings** in WSO2 DSS.
5. Click **Next** to open the **Operations** screen.

**Define an operation to invoke the query**

Data service operations are written to invoke queries.

1. Click **Add New Operation** to open **Add New Operation** screen.
2. Add a name for your operation in the **Operation Name** field.
3. In the **Query ID** field, select the query that you defined previously.
4. Save the operation.

**Finish creating the data service**

Once you have defined the operation, click **Finish** to complete the data service creation process. You will now be taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

**Uploading a Created Data Service**

**About Data Services and Resources** explain how to create a data service (.dbs file) file manually from scratch. After the .dbs file is created, deploy it to a running Data Services Server instance through the management console as explained below:

1. Log in to the management console and select **Upload** under **Data Service** menu.
2. Select the database backup (.dbs) file and click **Upload**. For example, you can upload one of the sample data services that are stored in the `<DSS_HOME>/repository/deployment/server/dataservices/samples/` folder.
3. If the file is deployed successfully, the **Deployed Services** window appears with the new data service listed. From here, you can manage your service. See **Managing Data Services**.

Alternatively, copy the file to `<DSS_HOME>/repository/deployment/server/dataservices` folder. It will be deployed instantly as hot deployment, which is enabled in Data Services Server by default.

**Generating a Data Service**

This option helps create data services automatically using a given database structure. When generating a data service, the server takes its table structure according to the structure specified in the datasource and automatically creates the **SELECT, INSERT, UPDATE and DELETE** operations.

- **Step 1: Setting up a datasource**
- **Step 2: Creating a Carbon datasource**
- **Step 3: Generating a data service**
- **Step 4: Verify the generated CRUD operations (Optional)**
- **Step 5: Invoking your data service**

---

**Step 1: Setting up a datasource**

Follow the steps given below to set up a MySQL database for this tutorial.

1. Install the MySQL server.
2. Download the JDBC driver for MySQL from [here](#) and copy it to your `<DSS_HOME>/repository/components/lib` directory.
3. Create the following database: **AccountDetails**
4. Create the following table:
CREATE TABLE ACCOUNT(AccountID int NOT NULL, Branch varchar(255) NOT NULL, AccountNumber varchar(255), AccountType ENUM('CURRENT', 'SAVINGS') NOT NULL, Balance FLOAT, ModifiedDate DATE, PRIMARY KEY (AccountID));

5. Enter the following data into the table:

```sql
INSERT INTO ACCOUNT VALUES
(1,"AOB","A00012","CURRENT",231221,'2014-12-02');
```

### Step 2: Creating a Carbon datasource

WSO2 Data Services Server comes with the datasource management feature, which allows users to create any RDBMS datasource or custom datasource that is used by the server to connect to databases or external data stores. See [Managing Datasources](#) in the WSO2 Administration Guide for instructions on how to use this feature. Note that these pre-defined datasources can also be added to a data service as Carbon datasources.

Follow the steps given below.

1. Log into the management console of WSO2 DSS and click **Datasources** in the **Configure** tab. This will open the **Datasources** screen.
2. Click **Add Datasource** and enter the following details corresponding to the 'AccountDetails' database:

![New Datasource](image-url)

3. Save the datasource.

### Step 3: Generating a data service

Once the datasource is created, follow the steps below to generate a data service.

1. Click **Generate** in the **Data Service** menu.
2. In the **Carbon Datasource** field, select a datasource from the drop down list.
3. In the **Database Name** field, give the name of the database that you created.

   ![Screen shot](image)

   Note that in most cases the database name should be the same name used when creating the datasource. If you want to find the database name for the datasource you selected in step 2 above, go to the **Configure** tab and click **Datasources** in the navigator. You can then get the details (including the database name) of a datasource by clicking **View**.

   However, some datasources such as Oracle and H2 may not require the exact database name.

4. Click **Next** to open the **Customize Service Generation** screen. The tables in the AccountDetails database will be listed. We only have one table (ACCOUNT) in the database as shown below.

   ![Screen shot](image)

5. Select the **ACCOUNT** table and click **Next** to open the **Service Generation** screen.
6. You must select a service generation mode from the following two options:
   - **Single Service**: Creates a single data service for operations of all tables.
   - **Multiple Services**: Creates a service per table, which will contain isolated operations for each table.
   This is only applicable if you have selected multiple tables in the previous step.

7. Click **Next** to open the **Generated Services** screen. A data service by the name of ACCOUNT_DataService is now generated.

8. Click **Finish** to complete the data service creation process. You will now be taken to the **Deployed Services** screen, which shows all the data services deployed on the server.

---

**Step 4: Verify the generated CRUD operations (Optional)**

You can verify that the relevant queries and CRUD operations are generated accurately:

1. Click the **ACCOUNT_DataService** service to open the service dashboard.
2. Click **Edit Data Service** to open the service in the **Create New Data Service** wizard.
3. Click **Next** until you get to the **Queries** screen. See the queries corresponding to CRUD operations are automatically generated:

<table>
<thead>
<tr>
<th>Query</th>
<th>Type</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert ACCOUNT_query</td>
<td>default</td>
<td><img src="image" alt="Edit Query" /> <img src="image" alt="Delete Query" /></td>
</tr>
<tr>
<td>update ACCOUNT_query</td>
<td>default</td>
<td><img src="image" alt="Edit Query" /> <img src="image" alt="Delete Query" /></td>
</tr>
<tr>
<td>select ACCOUNT_query</td>
<td>default</td>
<td><img src="image" alt="Edit Query" /> <img src="image" alt="Delete Query" /></td>
</tr>
<tr>
<td>delete ACCOUNT_query</td>
<td>default</td>
<td><img src="image" alt="Edit Query" /> <img src="image" alt="Delete Query" /></td>
</tr>
<tr>
<td>select with key ACCOUNT_query</td>
<td>default</td>
<td><img src="image" alt="Edit Query" /> <img src="image" alt="Delete Query" /></td>
</tr>
</tbody>
</table>

4. Click **Next** to get to the **Operations** screen. See the CRUD operations connected to the queries:
Step 5: Invoking your data service

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click the **Try this service** link for the **AccountDetails** data service. The TryIt Tool will open with the data service.
3. See that the CRUD operations are generated for the datasource.
4. Execute the CRUD operations from the TryIt tool to see the result.

Exposing Data as a REST Resource

WSO2 Data Services Server (WSO2 DSS) allows you to expose data stored in various datasources as REST-style resources in addition to SOAP services. The tutorials on creating data services from scratch will guide you on how to expose data as SOAP services.

The following instructions explain how you can expose data as a REST resource:

- Defining a REST resource
- Invoking the REST resource

Defining a REST resource

Let's take a data service that is already created using an RDBMS and expose the data as REST-style resources.

1. Follow the tutorial on exposing an RDBMS as a data service to create the **RDBMS** data service.
2. Log in to the product’s management console and click **List** under the **Main** menu. The **RDBMS** data service you created in the previous step will be listed.
3. Click the data service to open the dashboard for that data service.
4. Click **Edit Data Service** to open the data service using the **Create Data Service** wizard.
5. Click **Next**, until you get to the **Resources** screen.
6. Click **Add New Resource** to open the **Resources** screen shown below. You can now expose the data as REST resource.
The fields in the above screen are explained below.

- **Resource Path**: The resource name that is appended to the end of the resource URI. There are two ways of giving the resource path:
  - Use a query path with the following format: `<resource_path_name>/(Input_Parameter)`
  - Use a query parameter by giving a name for the resource path. For example, enter Product, if you are querying for products.
- **Resource Method**: The HTTP operation (GET, POST, PUT or DELETE).
- **Query ID**: The corresponding query for the resource invocation.

7. We will proceed to create a resource using the GET method. Enter values as follows:
   1. Enter GetAccount/{AccountID} in the **Resource Path** field.
   2. Select **GET** as the **Resource Method**.
   3. Select the query in the **Query ID** field.

8. Save the resource.
9. Save the data service.

**Invoking the REST resource**

The service can be invoked in REST-style via curl (http://curl.haxx.se). Shown below is the curl command to invoke the GET resource:

```
curl -X GET http://localhost:9763/services/RDBMS.HTTPEndpoint/GetAccount/1
```
Invoking Batch Operations

The request box feature allows you to invoke multiple operations (consecutively) to a datasource using a single batch operation. Follow the steps given below to define a data service that can invoke batch operations:

- Setting up a datasource
- Define a data service to invoke batch operations
- Invoking the data service

Setting up a datasource

Follow the steps given below to set up a MySQL database for this tutorial.

1. Install the MySQL server.
2. Download the JDBC driver for MySQL from here and copy it to your <DSS_HOME>/repository/components/lib directory.
3. Create the following database: Company
4. Create the following tables:
   - Offices table:
     ```
     CREATE TABLE `OFFICES` (`OfficeCode` int(11) NOT NULL, `AddressLine1` varchar(255) NOT NULL, `AddressLine2` varchar(255) DEFAULT NULL, `City` varchar(255) DEFAULT NULL, `State` varchar(255) DEFAULT NULL, `Country` varchar(255) DEFAULT NULL, `Phone` varchar(255) DEFAULT NULL, PRIMARY KEY (`OfficeCode`));
     ```
   - Employees table:
     ```
     CREATE TABLE `EMPLOYEES` (`EmployeeNumber` int(11) NOT NULL, `FirstName` varchar(255) NOT NULL, `LastName` varchar(255) DEFAULT NULL, `Email` varchar(255) DEFAULT NULL, `JobTitle` varchar(255) DEFAULT NULL, `OfficeCode` int(11) NOT NULL, PRIMARY KEY (`EmployeeNumber`, `OfficeCode`), CONSTRAINT `employees_ibfk_1` FOREIGN KEY (`OfficeCode`) REFERENCES `OFFICES` (`OfficeCode`));
     ```
5. Insert the following data into the Offices table:
Define a data service to invoke batch operations

Let's create a data service using the Create Data Service wizard:

1. Log into the management console of WSO2 DSS and click Create in the Data Service menu.
2. Add a name for the data service and select the Enable Boxcarring check box as shown below.

```
Data Service Name*  datascrvice
Data Service Namespace
Description

Advanced Configurations

Enable Batch Requests
Enable Boxcarring
Disable Legacy Boxcarring Mode
```

3. Click Next and add a new datasource.
4. Connect to the Company database that you defined above.
5. Click Next to go to the Queries screen.
6. Click Add New Query to specify the query details:
   1. Enter addEmployeeQuery as the query ID.
   2. Enter the following SQL dialect:

      ```
      insert into EMPLOYEES (employeeNumber, lastName, firstName, email, officecode) 
      values (:employeeNumber, 'test', 'test', :email, :officecode)
      ```

7. Click Generate Input Mapping and input mappings will be generated automatically:

```
<table>
<thead>
<tr>
<th>Input Mapping</th>
<th>Parameter Type</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>employeeNumber</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
<tr>
<td>email</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
<tr>
<td>officecode</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
</tbody>
</table>
```

8. Create another query for the Company datasource:
1. Enter `selectOfficebyEmployee` as the query ID.

2. Enter the following SQL dialect:

   ```sql
   select
       officecode, addressline1, addressline2, city, state, country, phone
   from OFFICES where officecode=?
   ```

9. Generate input and output mappings:

1. Click **Generate Input Mapping** and input mappings will be generated automatically for the office code field:

   ![Input Mappings](image)

2. Click **Generate Response** to automatically generate the output mappings for the fields in the `Offices` table:

   ![Result (Output Mapping)](image)

10. Create an operation for the `addEmployeeQuery` query as shown below.
11. Create another operation to invoke the `selectOfficebyEmployee` query:

```
Operations

<table>
<thead>
<tr>
<th>Operation Name*</th>
<th>selectOfficebyEmpOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Query ID*</td>
<td>selectOfficebyEmployee</td>
</tr>
</tbody>
</table>

**Operation Parameters**

```
<table>
<thead>
<tr>
<th>Query Parameter Name</th>
<th>Operation Parameter Name</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>param0</td>
<td>param0</td>
<td>Edit Delete</td>
</tr>
</tbody>
</table>
```

12. Click **Finish** to navigate to the **Deployed Services** window, from where you can manage data services.

---

**Invoking the data service**

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click **Try this Service** to open the data service from the TryIt tool. You will find that there is an additional request box operation
3. Select the batch operation and enter the following values:
1. Enter values for the **addEmployee** query.

2. Enter the same office code you used for the above query in the **selectOffice** operation:

   ```xml
   <p:request_box xmlns:p="http://ws.wso2.org/dataservice">
     <!--Exactly 1 occurrence-->
     <addEmployeeQueryOp xmlns="http://ws.wso2.org/dataservice">
       <!--Exactly 1 occurrence-->
       <xs:employeeNumber xmlns:xs="http://ws.wso2.org/dataservice">110</xs:employeeNumber>
       <!--Exactly 1 occurrence-->
       <xs:email xmlns:xs="http://ws.wso2.org/dataservice">test@email.com</xs:email>
       <!--Exactly 1 occurrence-->
     </addEmployeeQueryOp>
     <selectOfficebyEmpOp xmlns="http://ws.wso2.org/dataservice">
       <!--Exactly 1 occurrence-->
       <xs:param0 xmlns:xs="http://ws.wso2.org/dataservice">2</xs:param0>
     </selectOfficebyEmpOp>
   </p:request_box>
   
4. Execute the request box operation.

   1. First, the **addEmployee** operation will be executed. The data will be successfully added to the database.

   2. Secondly, the **selectOffice** operation will be executed and the details of the office attached to the employee will be returned in the result as shown below.

   ```xml
   <Entries xmlns="http://ws.wso2.org/dataservice">
     <Entry>
       <officecode>2</officecode>
       <addressline1>72</addressline1>
       <addressline2>Rose Street</addressline2>
       <city>Pasadena</city>
       <state>California</state>
       <country>United States</country>
       <phone>+152346343</phone>
     </Entry>
   </Entries>
   
**Invoking an Operation with Multiple Records**

The batch requests feature allows you to send multiple (IN-Only) requests to a datasource using a single operation (batch operation). Follow the steps given below to define a data service that can invoke batch requests:
Setting up a datasource

Follow the steps given below to set up a MySQL database for this tutorial.

1. Install the MySQL server.
2. Download the JDBC driver for MySQL from here and copy it to your `<DSS_HOME>/repository/components/lib` directory.
3. Create the following database: Company
4. Create the Employee table:

```
CREATE TABLE `EMPLOYEES` (`EmployeeNumber` int(11) NOT NULL, `FirstName` varchar(255) NOT NULL, `LastName` varchar(255) DEFAULT NULL, `Email` varchar(255) DEFAULT NULL, `JobTitle` varchar(255) DEFAULT NULL, `OfficeCode` int(11) NOT NULL, PRIMARY KEY (`EmployeeNumber`,`OfficeCode`), CONSTRAINT `employees_ibfk_1` FOREIGN KEY (`EmployeeNumber`,`OfficeCode`) REFERENCES `OFFICES` (`OfficeCode`));
```

Define a data service to insert records in batches

Let's create a data service using the Create Data Service wizard:

1. Log into the management console of WSO2 DSS and click Create in the Data Service menu.
2. Add a name for the data service and select the Enable Batch Requests check box as shown below.
3. Click Next and add a new datasource.
4. Connect to the Company database that you defined above.
5. Click Next to open the Queries screen.
6. Click Add New Query to specify the query details:

   1. Enter `addEmployeeQuery` as the query ID.
   2. Enter the following SQL dialect:
7. Click **Generate Input Mapping** and an input mapping will be generated automatically for the `employeeNumber`, `email`, and `OfficeCode` field.

![Input Mappings Table]

8. Create an operation for the `addEmployeeQuery` query as shown below.

![Operations Table]

9. Click **Finish** to navigate to the **Deployed Services** window, from where you can manage data services.

**Invoking the data service**

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click **Try this Service** to open the data service from the TryIt tool.
3. Click **Try this Service** to open the data service from the TryIt tool. There will be a `batch_request` operation automatically generated.
4. Select the batch operation.
5. Enter multiple transactions as shown below. In this example, we are sending two transactions.
6. Execute the batch operation. You will find that all the records have been inserted into the database simultaneously.

**Defining Nested Queries**

Nested queries help you to use the result of one query as an input parameter of another, and the queries executed in a nested query work in a transactional manner. Follow the steps given below to add a nested query to a data service.

- Setting up a datasource
- Writing a nested query for a datasource
- Invoking the data service

### Setting up a datasource

Follow the steps given below to set up a MySQL database for this tutorial.

1. Install the MySQL server.
2. Download the JDBC driver for MySQL from here and copy it to your `<DSS_HOME>/repository/components/lib` directory.
3. Create the following database: Company
4. Create the following tables:
• **Offices** table:

```sql
CREATE TABLE `OFFICES` (`OfficeCode` int(11) NOT NULL,
`AddressLine1` varchar(255) NOT NULL, `AddressLine2` varchar(255)
DEFAULT NULL, `City` varchar(255) DEFAULT NULL, `State` varchar(255)
DEFAULT NULL, `Country` varchar(255) DEFAULT NULL, `Phone` varchar(255)
DEFAULT NULL, PRIMARY KEY (`OfficeCode`));
```

• **Employees** table:

```sql
CREATE TABLE `EMPLOYEES` (`EmployeeNumber` int(11) NOT NULL,
`FirstName` varchar(255) NOT NULL, `LastName` varchar(255)
DEFAULT NULL, `Email` varchar(255) DEFAULT NULL, `JobTitle` varchar(255)
DEFAULT NULL, `OfficeCode` int(11) NOT NULL, PRIMARY KEY (`EmployeeNumber`,
`OfficeCode`), CONSTRAINT `employees_ibfk_1` FOREIGN KEY (`OfficeCode`) REFERENCES
`OFFICES` (`OfficeCode`));
```

5. Insert the following data into the tables:

• **Add to the Offices table:**

```sql
INSERT INTO OFFICES VALUES (1,"51","Glen Street","Norwich","London","United Kingdom","+441523624");
INSERT INTO OFFICES VALUES (2,"72","Rose Street","Pasadena","California","United States","+152346343");
```

• **Add to the Employees table:**

```sql
INSERT INTO EMPLOYEES VALUES (1,"John","Gardiner","john@office1.com","Manager",1);
INSERT INTO EMPLOYEES VALUES (2,"Jane","Stewart","jane@office2.com","Head of Sales",2);
INSERT INTO EMPLOYEES VALUES (3,"David","Green","david@office1.com","Manager",1);
```

You will now have two tables in the **Company** database as shown below:

• **Offices** table:

<table>
<thead>
<tr>
<th>OfficeCode</th>
<th>AddressLine1</th>
<th>AddressLine2</th>
<th>City</th>
<th>State</th>
<th>Country</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51</td>
<td>Glen Street</td>
<td>Norwich</td>
<td>London</td>
<td>United Kingdom</td>
<td>+441523624</td>
</tr>
<tr>
<td>2</td>
<td>72</td>
<td>Rose Street</td>
<td>Pasadena</td>
<td>California</td>
<td>United States</td>
<td>+152346343</td>
</tr>
</tbody>
</table>

• **Employees** table:
Writing a nested query for a datasource

Let's create a data service using the Create Data Service wizard:

1. Log into the management console of WSO2 DSS and click Create in the Data Service menu.
2. Add a name for the data service and go to the next step.
3. Connect to the Company database that you defined above.
4. Click Next to open the Queries screen.
5. Click Add New Query to specify the query details:
   1. Enter EmployeeOfficeSQL as the query ID.
   2. Enter the following SQL dialect:

```
select EmployeeNumber, FirstName, LastName, Email, JobTitle, OfficeCode from EMPLOYEES where OfficeCode=:OfficeCode
```

6. Generate input and output mappings:
   1. Click Generate Input Mapping and an input mapping will be generated automatically for the OfficeCode field:

```
<table>
<thead>
<tr>
<th>Mapping Name</th>
<th>Parameter Type</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>OfficeCode</td>
<td>SCALAR</td>
<td>STRING</td>
<td></td>
</tr>
</tbody>
</table>
```

   2. Click Generate Response to automatically generate output mappings for the EmployeeNumber, FirstName, LastName, Email, Job Title, and Office Code fields.

7. Start creating a new query for the Company datasource:
1. Enter `listOfficeSQL` as the query ID.

2. Enter the following SQL dialect:

   ```sql
   select OfficeCode, AddressLine1, AddressLine2, City, State, Country, Phone from OFFICES
   ```

8. Click **Generate Response** to create output mappings for the fields specified in the above query.

9. Create a new output mapping.

   1. Select **Query** for the **Mapping Type** field.
   
   2. Enter the `employeeOfficeSQL` query you created previously in the **Select Query** field.
   
   3. Save the output mapping
10. Save the listOfficeSQL query.

11. Create an operation for the listOfficeSQL query as shown below.

12. Save the listOfficeSQLOP operation.

13. Click Finish to navigate to the Deployed Services window, from where you can manage data services.

**Invoking the data service**

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the Deployed Services screen.
2. Click Try this Service to open the data service from the TryIt tool.
3. Execute the listOfficeSQLOP operation and you will see the following result:
<Entries xmlns="http://ws.wso2.org/dataservice">
    <Entry>
        <OfficeCode>1</OfficeCode>
        <AddressLine1>51</AddressLine1>
        <AddressLine2>Glen Street</AddressLine2>
        <City>Norwich</City>
        <State>London</State>
        <Country>United Kingdom</Country>
        <Phone>+441523624</Phone>
        <Entries>
            <Entry>
                <EmployeeNumber>1</EmployeeNumber>
                <FirstName>John</FirstName>
                <LastName>Gardiner</LastName>
                <Email>john@office1.com</Email>
                <JobTitle>Manager</JobTitle>
                <OfficeCode>1</OfficeCode>
            </Entry>
            <Entry>
                <EmployeeNumber>3</EmployeeNumber>
                <FirstName>David</FirstName>
                <LastName>Green</LastName>
                <Email>david@office1.com</Email>
                <JobTitle>Manager</JobTitle>
                <OfficeCode>1</OfficeCode>
            </Entry>
        </Entries>
    </Entry>
    <Entry>
        <OfficeCode>2</OfficeCode>
        <AddressLine1>72</AddressLine1>
        <AddressLine2>Rose Street</AddressLine2>
        <City>Pasadena</City>
        <State>California</State>
        <Country>United States</Country>
        <Phone>+152346343</Phone>
        <Entries>
            <Entry>
                <EmployeeNumber>2</EmployeeNumber>
                <FirstName>Jane</FirstName>
                <LastName>Stewart</LastName>
                <Email>jane@office2.com</Email>
                <JobTitle>Head of Sales</JobTitle>
                <OfficeCode>2</OfficeCode>
            </Entry>
        </Entries>
    </Entry>
</Entries>

Receiving Notifications from Data Services
Eventing support is provided by the WS-Eventing Web services standard. When a data service request or response triggers an event, the subscribers listening to those events receive notifications. The criteria for triggering an event as well as the destination to which the event notifications should be sent are defined per data service query. When a certain event-trigger is activated, emails will be sent to all the respective subscribers.

You can create an event trigger from a query as explained below.

- **Before you begin**
- **Enabling notifications for a query in a data service**
- **Invoking the data service**

### Before you begin
- Update email configurations: Open the `axis2_client.xml` file from the `<DSS_HOME>/repository/conf/axis2` folder and add the following XML element:

```xml
<transportSender name="mailto"
    class="org.apache.axis2.transport.mail.MailTransportSender">
    <parameter name="mail.smtp.from">esb.sample@gmail.com</parameter>
    <parameter name="mail.smtp.user">esb.sample</parameter>
    <parameter name="mail.smtp.password">esb*sample8</parameter>
    <parameter name="mail.smtp.host">smtp.gmail.com</parameter>
    <parameter name="mail.smtp.port">587</parameter>
    <parameter name="mail.smtp.starttls.enable">true</parameter>
    <parameter name="mail.smtp.auth">true</parameter>
</transportSender>
```

- Set up a datasource:
  1. Install the MySQL server.
  2. Download the JDBC driver for MySQL from [here](#) and copy it to your `<DSS_HOME>/repository/components/lib` directory.
  3. Create the following database: Company
  4. Create the ACCOUNT table in the Company database:

```sql
CREATE TABLE ACCOUNT(AccountID int NOT NULL, Branch varchar(255) NOT NULL, AccountNumber varchar(255) NOT NULL, AccountType ENUM('CURRENT', 'SAVINGS') NOT NULL, Balance FLOAT, ModifiedDate DATE, PRIMARY KEY (AccountID));
```

  5. Enter the following data into the ACCOUNT table:

```sql
INSERT INTO ACCOUNT VALUES (1,"AOB","A00012","CURRENT",231221,'2014-12-02');
```

### Enabling notifications for a query in a data service
Let's create a data service using the Create Data Service wizard:

1. Log into the management console of WSO2 DSS and click Create in the Data Service menu.
2. Add a name for the data service and go to the next step.
3. Connect to the Company database that you defined above.
4. Click Next to go to the Queries screen.
5. Click Add New Query to specify the query details:
   1. Enter UpdateAccBalance as the query ID.
      2. Enter the following SQL dialect:

```
UPDATE ACCOUNT SET Balance=:Balance WHERE AccountID=:AccountID
```

6. Click Generate Input Mapping to automatically generate input mappings for the AccountID and Balance fields.

7. At the bottom of the page you will find the Events section:

8. Click Manage Events and add a new event as shown below:
The fields in the **Add Events** screen are explained below.

1. **Event Id**: account_balance_low_trigger

   The ID used for identifying the event-trigger used in data services queries.

2. **Xpath**: /updateAccountBalanceQ/Balance<200

   Represents an XPath expression that is run against the XML message presented. That is, the request/response message. When this evaluation returns `true`, the event is triggered.

3. **Target Topic**: product_stock_low_topic

   The topic to which the event notifications are published.

4. **Event Sink URL**: mailto:youremail@wso2.com

   A subscription can be any endpoint that is complaint with WS-Eventing. For example, you can use an SMTP transport to send a message to a mail inbox, where an email address is given as the subscription. Here, many subscriptions can be defined for the given topic.

9. Save the event.

10. Save the **UpdateAccBalance** query.

11. Create an operation for the **UpdateAccBalance** query as shown below.
12. Save the Operation.

13. Click **Finish** to navigate to the **Deployed Services** window from where you can manage data services.

---

### Invoking the data service

You can try the data service you created by using the TryIt tool that is in your product by default.

1. Go to the **Deployed Services** screen.
2. Click **Try this Service** to open the data service from the TryIt tool.
3. You will find the new operation, which you have created.
4. Enter a value less than 200 as the balance and click on **Send**.
5. You should receive an email when this happens.

### Handling Distributed Transactions

WSO2 Data Services Server supports data federation, which means that a single data service can expose data from multiple datasources. However, if you have multiple RDBMSs connected to your data service, and if you need to perform IN-ONLY operations (operations that can insert data and modify data in the datasource) in a coordinated manner, the RDBMSs need to be defined as XA datasources. Consider a scenario where you have two MySQL databases. You can define a single data service for these databases and insert data into both as explained below.

- Setting up distributed MySQL databases
- Adding the datasources to a data service
- Defining queries for the datasources
- Inserting data into the distributed RDBMs

---

#### Setting up distributed MySQL databases

Follow the steps given below to set up a MySQL database for this tutorial.

1. Install the MySQL server.
2. Download the JDBC driver for MySQL from here and copy it to your `<DSS_HOME>/repository/components/lib` directory.

3. Set up a database for storing information of offices:
   1. Create a database called `OfficeDetails`.
   2. Create the `Offices` table:
      ```
      CREATE TABLE `OFFICES` (
          `OfficeCode` int(11) NOT NULL,
          `AddressLine1` varchar(255) NOT NULL, `AddressLine2` varchar(255) DEFAULT NULL, `City` varchar(255) DEFAULT NULL, `State` varchar(255) DEFAULT NULL, `Country` varchar(255) DEFAULT NULL, `Phone` varchar(255) DEFAULT NULL, PRIMARY KEY (`OfficeCode`));
      ```

4. Set up a database for storing information of employees:
   1. Create a database called `EmployeeDetails`.
   2. Create the `Employees` table:
      ```
      CREATE TABLE `EMPLOYEES` (
          `EmployeeNumber` int(11) NOT NULL, `FirstName` varchar(255) NOT NULL, `LastName` varchar(255) DEFAULT NULL, `Email` varchar(255) DEFAULT NULL, `JobTitle` varchar(255) DEFAULT NULL, `OfficeCode` int(11) NOT NULL, PRIMARY KEY (`EmployeeNumber`));
      ```

Adding the datasources to a data service

Let's create a data service using the Create Data Service wizard:

1. Log into the management console of WSO2 DSS and click Create in the Data Service menu.
2. Add a name for the data service and go to the next step.
3. Select the Enable Boxcarring check box.
4. Click Next to go to the Datasources screen.
5. Create a datasource connection for the `CompanyDetails` database follows:
   1. Enter XAoffices as the datasource ID.
   2. Select RDBMS as the datasource type.
   3. Select External Datasource from the adjoining field.
   4. Select MySQL as the database engine.
   5. Enter the following database class corresponding to MySQL:
      ```java
      com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
      ```
   6. Click Add Properties and create the connection settings for the `OfficeDetails` database.
      ```
      URL: jdbc:mysql://localhost:3306/OfficeDetails
      USER: root
      ```
6. Save the **XAoffices** datasource.

7. Create a datasource connection for the **EmployeeDetails** database as follows:
   1. Enter **XAemployees** as the datasource ID.
   2. Select **RDBMS** as the datasource type.
   3. Select **External Datasource** from the adjoining field.
   4. Select **MySQL** as the database engine.
   5. Enter the following database class corresponding to MySQL:
      
      ```java
      com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
      ```
   6. Click **Add Properties** and create the connection settings for the **OfficeDetails** database.
      
      - URL: `jdbc:mysql://localhost:3306/EmployeeDetails`
      - USER: root

8. Save the **XAemployees** datasource.

---

**Defining queries for the datasources**

1. Click **Add New Query** to specify an insert query for the **XAoffices** datasource:
   1. Enter **InsertOfficeQuery** as the query ID.
   2. Enter the following SQL dialect:
1. Click **Generate Input Mapping** and an input mapping will be generated automatically for the fields in the datasource:

<table>
<thead>
<tr>
<th>Mapping Name</th>
<th>Parameter Type</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>OfficeCode</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
<tr>
<td>AddressLine1</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
</tbody>
</table>

2. Save the query.
3. Click **Add New Query** to specify an insert query for the **XAemployees** datasource:
   1. Enter **InsertEmployeeQuery** as the query ID.
   2. Enter the following SQL dialect:

   `insert into EMPLOYEES (employeeNumber, firstName, lastName, email, jobtitle, OfficeCode) values(:employeeNumber, :firstName, :lastName, 'test@gmail.com', 'test', :OfficeCode)`

4. Click **Generate Input Mapping** and an input mapping will be generated automatically for the fields in the datasource:

<table>
<thead>
<tr>
<th>Mapping Name</th>
<th>Parameter Type</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>employeeNumber</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
<tr>
<td>firstName</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
<tr>
<td>lastName</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
<tr>
<td>OfficeCode</td>
<td>SCALAR</td>
<td>STRING</td>
<td>Edit</td>
</tr>
</tbody>
</table>

5. Save the query.
6. Click **Next** to go to the **Operations** section. Define operations to invoke the two queries defined above.
   - Create the **InsertOfficeOp** for the **InsertOfficeQuery**.
   - Create the **InsertEmployeeOp** for the **InsertEmployeeQuery**.
7. Finish creating the data service.

### Inserting data into the distributed RDBMs

1. Go to the **Deployed Services** page and you will see the data service listed.
2. Click **Try this service** to open the TryIt tool.
3. Select the batch operation that is created by default (request_box_operation).
4. Specify the values that should be inserted to the **OfficeDetails** database and **EmployeeDetails** database respectively.
5. Invoke the operation.
6. See that the data is successfully inserted into the two databases.
7. Now, enter another set of values for the two operations. However, you can enter an erroneous value for one field.
8. Invoke the operation.
9. See that no records have been entered into either database.

**Exposing Data as an OData Service**

In this tutorial, we will run through the process of exposing and RDBMS as an OData service. When OData is enabled for a datasource, you do not need to manually define CRUD operations. These will be automatically created.

- Setting up an RDBMS
- Expose the RDBMS as an OData service
- Access the data service using CRUD operations

Note that the OData feature can only be used for RDBMS and Cassandra datasources.

---

### Setting up an RDBMS

Follow the steps given below to set up a MySQL database for this tutorial.

1. Install the MySQL server.
2. Download the JDBC driver for MySQL from [here](#) and copy it to your `<DSS_HOME>/repository/components/lib` directory.
3. Create a MySQL database with the following table:

   ```
   CREATE TABLE ACCOUNT(AccountID int NOT NULL,Branch varchar(255) NOT NULL, AccountNumber varchar(255),AccountType ENUM('CURRENT', 'SAVINGS') NOT NULL, Balance FLOAT,ModifiedDate DATE,PRIMARY KEY (AccountID));
   ```

4. Enter the following data into the table:

   ```
   INSERT INTO ACCOUNT VALUES
   (1,"AOB","A00012","CURRENT",231221,'2014-12-02');
   ```

### Expose the RDBMS as an OData service

Follow the steps given below.

1. Log in to the management console and select **Create** under **Data Service** menu.
2. Create a data service to expose the MySQL database you created above.
3. When you connect the datasource, select the **OData** check box as shown below.
4. Click Finish to save the data service.

5. Go to the Deployed Services screen and click the data service that you created. The endpoints for accessing data in the datasource will be shown as follows:

### Endpoints

- http://10.100.5.65:9763/services/dataservice/
- https://10.100.5.65:9443/services/dataservice/
- local:///services/dataservice/

---

Access the data service using CRUD operations

Open a command prompt execute the following CURL commands using CRUD operations:

- To get the service document:

  ```bash
curl -X GET -H 'Accept: application/json'
  https://localhost:9443/odata/{data_service_name}/{data_source_id}
  ```

- To get the metadata of the service:

  ```bash
curl -X GET -H 'Accept: application/xml'
  https://localhost:9443/odata/{data_service_name}/{data_source_id}/$metadata
  ```

- To read details from the ACCOUNT table:

  ```bash
curl -X GET -H 'Accept: application/xml'
  https://localhost:9443/odata/{data_service_name}/{data_source_id}/ACCOUNT
  ```
Scheduling Tasks

Task scheduling is used to invoke a data service operation periodically or for a specified number of times. Read about data services and operations. The scheduling functionality is useful when a specific data service operation scheduled for execution 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, we can schedule a task on `getProductQuantity` operation and set an event to send an email if the quantity goes down to some level.

Task scheduling functionality is provided by the following feature in the WSO2 feature repository:

- **Name**: Data Service Tasks Feature
- **Identifier**: org.wso2.carbon.dataservices.task.feature.group

The following topics are covered:

- Configuring server for task handling
- Adding Scheduled Tasks

Configuring server for task handling

For instructions on how to configure the Task Scheduling component for your server, see the admin guide.

Adding Scheduled Tasks

Follow the steps given below to schedule a task.

1. Log in to the management console and click Data Services > Scheduled Tasks in the Main menu. The Scheduled Tasks window opens.
2. Click Add New Task to open the New Scheduled Task screen.
3. Enter a name for the task in the Task Name field.
4. Specify how the task should be triggered using the following fields:
   - **Task Repeat Count**: Number of cycles to be executed. If you enter 0, the task will execute once. If you enter 1, the task will execute twice and so on.
   - **Task Interval**: Time gap between two consecutive task executions
   - **Start Time**: Starting time of the scheduled task. If this is not given, the task will start as soon as it is scheduled.
5. Select one of the following values for the Scheduling Type field:
   - **DataService Operation**: If this option is selected, the task will be invoking a data service operation.
   - **DataService Task Class**: If this option is selected, the task will be using a custom class that implements the `org.wso2.carbon.dataservices.taskDataTask` interface.
6. To successfully create a task, provide the following set of properties:
• If the scheduling type is **DataService Operation**, the following fields should be used:

1. **Data Service Name**: Name of the relevant data service.
2. **Operation Name**: Data service operation to be executed from the task.

Note that only data services with HTTP endpoints are available when scheduling tasks to invoke data service operations. Also, you can use only operations with no input parameters when scheduling.

• If the scheduling type is **DataService Task Class**, you must specify the Java class that implements the `org.wso2.carbon.dataservices.task.DataTask` interface in the **DataService Task Class** field.

The definition of the interface is as follows:

```java
package org.wso2.carbon.dataservices.task;

/**<*
 * This interface represents a scheduled data task.
 */
public interface DataTask
    void execute(DataTaskContext ctx);
}
```

The following code snippet shows a sample DataTask implementation:
package samples;
import java.util.HashMap;
import java.util.Map;
import org.wso2.carbon.dataservices.core.DataServiceFault;
import org.wso2.carbon.dataservices.core.engine.ParamValue;
import org.wso2.carbon.dataservices.task.DataTask;
import org.wso2.carbon.dataservices.task.DataTaskContext;

public class SampleDataTask implements DataTask {
    @Override
    public void execute(DataTaskContext ctx) {
        Map<String, ParamValue> params = new HashMap<String, ParamValue>();
        params.put("increment", new ParamValue("1000"));
        params.put("employeeNumber", new ParamValue("1002"));
        try {
            ctx.invokeOperation("RDBMSSample", "incrementEmployeeSalary", params);
        } catch (DataServiceFault e) {
            // handle exception
        }
    }
}
Deep Dive

See the following topics for in-depth information on how to use WSO2 Data Services Server.

- Installation Guide
- Running the product on a Preferred Profile
- Product Administration
- Java Documentation
- Working with Data Services
- Working with DSS Tooling

Installation Guide

The section provides instructions to download, install, run, and get started quickly with WSO2 Data Services Server.

- Downloading the Product
- Installation Prerequisites
- Installing the Product
- Running the Product

Downloading the Product

Follow the instructions below to download the product. You can also download and build the source code.

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

The binary distribution contains the binary files for both MS Windows and Linux-based operating systems, compressed into a single ZIP file. This distribution is recommended for most users.

After downloading the binary distribution, 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

<table>
<thead>
<tr>
<th>Memory</th>
<th>~ 2 GB minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>~ 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.</td>
</tr>
</tbody>
</table>
Disk

- ~ 1 GB, excluding space allocated for log files and databases.

Environment compatibility

Operating Systems / Databases

- All WSO2 Carbon-based products are Java applications that can be run on any platform that is Oracle JDK 7/8 compliant. Also, we do not recommend OpenJDK as we do not support it or test our products with it.
- All WSO2 Carbon-based products are generally compatible with most common DBMSs. For more information, see Working with Databases.
- It is not recommended to use Apache DS in a production environment due to issues with scalability. Instead, it is recommended to use an LDAP like OpenLDAP for user management.
- For environments that WSO2 products are tested with, see Compatibility of WSO2 Products.
- If you have difficulty in setting up any WSO2 product in a specific platform or database, please contact us.

Required applications

The following applications are required for running the DSS and its samples or for building from the source code. Mandatory installations are marked with *.

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th>Version</th>
<th>Download Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Java SE Development Kit (JDK)*</td>
<td>- To launch the product as each product is a Java application.</td>
<td>JDK 7 or 8.</td>
<td><a href="http://java.sun.com/javase/downloads/index.jsp">http://java.sun.com/javase/downloads/index.jsp</a></td>
</tr>
<tr>
<td></td>
<td>- To build the product from the source distribution (both JDK and Apache Maven are required).</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- To run Apache Ant.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle and IBM JRE 1.7 are also supported when running (not building) WSO2 products.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>We do not recommend OpenJDK as we do not support it or test our products with it.</td>
<td></td>
</tr>
</tbody>
</table>
### Apache ActiveMQ JMS Provider
- To enable the product's JMS transport and try out JMS samples. The ActiveMQ client libraries must be installed in the product's classpath before you can enable the JMS transport.
- 5.5.0 or later
- If you use any other JMS provider (e.g., Apache Qpid), install any necessary libraries and/or components.
- http://activemq.apache.org

### Apache Ant
- To compile and run the product samples.
- 1.7.0 or later
- http://ant.apache.org

### Git
- Download the source code and build the product from the source distribution.
- Linux - http://git-scm.com/download/linux

### Apache Maven
- To build the product from the source distribution (both JDK and Apache Maven are required). If you are installing by downloading and extracting the binary distribution instead of building from the source code, you do not need to install Maven.
- 3.0.*
- http://maven.apache.org

### Web Browser
- Required by all WSO2 products to access each product's Management Console. The Web Browser must be JavaScript enabled to take full advantage of the Management console.
- NOTE: On Windows Server 2003, you must not go below the medium security level in Internet Explorer 6.x.

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

- Installing on Linux or OS X
- Installing on Solaris
- Installing on Windows
- Installing as a Windows Service
Installing the Product

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

- Installing on Linux or OS X
- Installing on Solaris
- Installing on Windows
- Installing as a Windows Service
- Installing as a Linux Service

Installing on Linux or OS X

Follow the instructions below to install the required applications and WSO2 DSS on Linux or Mac OS X.

Install the required applications

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

Installing the product

1. If you have not done so already, download the latest version of DSS as described in Downloading the Product.
2. Extract the archive file to a dedicated directory for the product, which will hereafter be referred to as <DSS_HOME>.

Setting 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 favourite Linux text editor, such as vi, emacs, pico or mcedit.
2. 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:
3. Save the file.

If you do not know how to work with text editors in a Linux SSH session, run the following command:

```
cat >> .bashrc
```

Paste the string from the clipboard and press "Ctrl+D".

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

```
echo $JAVA_HOME
```

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.

### SUSE Linux

When using SUSE Linux, it ignores `/etc/resolv.conf` and only looks at the `/etc/hosts` file. This
You are now ready to run the product.

**Installing on Solaris**

Before you begin, please see our compatibility matrix to find out if this version of the product is fully tested on Solaris.

Follow the instructions below to install DSS on Solaris.

**Installing the required applications**

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

**Installing the DSS**

1. Download the latest version of the DSS as described in Downloading the Product.
2. Extract the archive file to a dedicated directory for the DSS, which will hereafter be referred to as `<DSS_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 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:
3. Save the file.

If you do not know how to work with text editors in an 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`

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

5. The system returns the JDK installation path.

### Setting system properties

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

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

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

You are now ready to run the product.

### Installing on Windows

**Before you begin, please see our compatibility matrix** to find out if this version of the product is fully tested on Windows.

Follow the instructions below to install DSS on Windows.

#### Installing the required applications

- Be sure your system meets the **Installation Prerequisites**. Java Development Kit (JDK) is essential to run the product.
• Be sure that the PATH environment variable is set to "C:\Windows\System32", because the findstr executable is stored in this path.

Installing the DSS

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

```bash
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.
- 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 <Y AJSW_HOME>/conf/ directory and in the <PRODUCT_HOME>/bin/yajsw/ directory of many WSO2 products. Following is the minimal wrapper.conf configuration for running a WSO2 product as a Windows service. Open your wrapper.conf file, set its properties as follows, and save it in <YAJSW_HOME>/conf/ directory.

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

You must manually add the following property to the wrapper.conf file to avoid errors in the management console:

```python
wrapper.java.additional.26 = -Dorg.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING=false
```

If this property is not added, you may come across an exception that will result in blank pages when you sign into the product management console.

### Minimal wrapper.conf configuration

```python
# 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
# Application main class or native executable
# One of the following properties MUST be defined
# Java Application main class
wrapper.java.app.mainclass=org.wso2.carbon.bootstrap.Bootstrap

# Log Level for console output. (See docs for log levels)
wrapper.console.loglevel=INFO

# Log file to use for wrapper output logging.
```
wrapper.logfile=${wrapper_home}\log\wrapper.log
# Format of output for the log file. (See docs for formats)
#wrapper.logfile.format=LPTM
# Log Level for log file output. (See docs for log levels)
#wrapper.logfile.loglevel=INFO
# Maximum size that the log file will be allowed to grow to before
# the log is rolled. Size is specified in bytes. The default value
# of 0, disables log rolling by size. May abbreviate with the 'k' (kB) or
# 'm' (mB) suffix. For example: 10m = 10 megabytes.
# If wrapper.logfile does not contain the string ROLLNUM it will be
automatically added as suffix of the file name
wrapper.logfile.maxsize=10m
# Maximum number of rolled log files which will be allowed before old
# files are deleted. The default value of 0 implies no limit.
wrapper.logfile.maxfiles=10
# Title to use when running as a console
wrapper.console.title="WSO2 Carbon"
# Wrapper Windows Service and Posix Daemon Properties
#********************************************************************
# Name of the service
wrapper.ntservice.name="WSO2CARBON"
# Display name of the service
wrapper.ntservice.displayname="WSO2 Carbon"
# Description of the service
wrapper.ntservice.description="Carbon Kernel"
# Wrapper System Tray Properties
#********************************************************************
# enable system tray
wrapper.tray = true
# TCP/IP port. If none is defined multicast discovery is used to find the
port
# Set the port in case multicast is not possible.
wrapper.tray.port = 15002
# Exit Code Properties
# Restart on non zero exit code
#********************************************************************
wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART
# Trigger actions on console output
#********************************************************************
# On Exception show message in system tray
wrapper.filter.trigger.0=Exception
wrapper.filter.script.0=scripts\trayMessage.gv
wrapper.filter.script.0.args=Exception
# genConfig: further Properties generated by genConfig
#********************************************************************
placeHolderSoGenPropsComeHere=
wrapper.java.command = ${java_home}\bin\java
wrapper.java.classpath.1 = ${java_home}\lib\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\bin\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 = -Xbootclasspath/a:${carbon_home}\lib\xboot\*.jar
wrapper.java.additional.2 = -Xms256m
wrapper.java.additional.3 = -Xmx1024m
wrapper.java.additional.4 = -XX:MaxPermSize=256m
wrapper.java.additional.5 = -XX:+HeapDumpOnOutOfMemoryError
wrapper.java.additional.6 = -XX:HeapDumpPath=${carbon_home}\repository\logs\heap-dump.hprof
wrapper.java.additional.7 = -Dcom.sun.management.jmxremote
wrapper.java.additional.8 = -Djava.endorsed.dirs=${carbon_home}\lib\endorsed;${java_home}\jre\lib\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root=/
wrapper.java.additional.10 = -Dcarbon.home=${carbon_home}
wrapper.java.additional.11 = -Dwso2.server.standalone=true
wrapper.java.additional.12 = -Djava.command=${java_home}\bin\java
wrapper.java.additional.13 = -Djava.io.tmpdir=${carbon_home}\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 =
WSO2 Data Services Server, version 3.2.2

Setting up CARBON_HOME

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

Running the product in console mode

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

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

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

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

```
runConsole.bat
```

For example:

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

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

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

```
installService.bat
```

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

```
startService.bat
```

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

```
stopService.bat
```

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

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

```
uninstallService.bat
```

The console will display a message confirming that the WSO2CARBON service was removed.
Installing as a Linux Service

Follow the sections below to run a WSO2 product as a Linux service:

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

**Prerequisites**

Install JDK 1.6.24 or later or 1.7.* and set up the JAVA_HOME environment variable.

**Setting up CARBON_HOME**

Extract the WSO2 product to a preferred directory in your machine and set the environment variable CARBON_HOME to the extracted 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 "$1 in
  start)
    echo "Starting the Service"
    ;;
  stop)
    echo "Stopping the Service"
    ;;
  restart)
    echo "Restarting the Service"
    ;;
  *)
    echo "$Usage: $0 {start|stop|restart}"
    exit 1
esac

Given below is a sample startup script. <PRODUCT_HOME> can vary depending on the WSO2 product’s directory.

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

startcmd='<PRODUCT_HOME>/bin/wso2server.sh start > /dev/null &'
restartcmd='<PRODUCT_HOME>/bin/wso2server.sh restart > /dev/null &'
stopcmd='<PRODUCT_HOME>/bin/wso2server.sh stop > /dev/null &'

case "$1" in
  start)
    echo "Starting the WSO2 Server ..."
    su -c "${startcmd}" user1
    ;;
  restart)
    echo "Re-starting the WSO2 Server ..."
    su -c "${restartcmd}" user1
    ;;
  stop)
    echo "Stopping the WSO2 Server ..."
    su -c "${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.
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 step 1:

```bash
sudo update-rc.d prodserver 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.

---

Running the Product

To run WSO2 products, you start the product server at the command line. You can then run the Management Console application to configure and manage the product. This page describes how to run the product in the following sections:

- Before you begin
- Starting the Server
- Accessing the Management Console
- Stop the Server

Before you begin

Before you start the server, note the following:

- The `config-validation.xml` file in the `<PRODUCT_HOME>/repository/conf/etc` directory contains a list of recommended system parameters, which are validated against your system when the server starts. See Configuring config-validation.xml for details on modifying these parameters before starting the server.
- The management console of WSO2 products uses the `org.apache.coyote.http11.Http11NioProtocol` Java connector by default. This connector is configured in the `catalina-server.xml` file stored in the `<PRODUCT_HOME>/repository/conf/tomcat` directory. This transport connector must be properly configured in this file for the management console to be accessible. See HTTP Servlet Transport for more information.

Starting the Server

Follow the relevant instructions based on the operating system you use.
On Windows/Linux/Mac OS

To start the server, you run the script `wso2server.bat` (on Windows) or `wso2server.sh` (on Linux/Mac OS) from the `bin` folder. Alternatively, you can install and run the server as a Windows service.

To start and stop the server in the background mode of Linux, run `wso2server.sh start` and `wso2server.sh stop` commands.

1. Open a command prompt:
   - On Windows, choose **Start -> Run**, type `cmd` at the prompt, and press Enter.
   - On Linux/Mac OS, establish a SSH connection to the server or log in to the text Linux console.
2. Execute one of the following commands, where `<IS_HOME>` is the directory where you installed the product distribution:
   - On Windows: `<IS_HOME>/bin/wso2server.bat --run`
   - On Linux/Mac OS: `sh <IS_HOME>/bin/wso2server.sh`

If you want to provide access to the production environment without allowing any user group (including admin) to log into the management console, execute one of the following commands.

- On Windows: `<PRODUCT_HOME>/bin/wso2server.bat --run -DworkerNode`
- On Linux/Mac OS: `sh <PRODUCT_HOME>/bin/wso2server.sh -DworkerNode`

For additional options you can use with these startup commands, type `-help` after the command, such as: `sh <PRODUCT_HOME>/bin/wso2server.sh -help`

The operation log appears. When the product server is running, the log displays the message "WSO2 Carbon started in 'n' seconds."

On Solaris

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

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

1. Click **Launch -> Run Applications**, type `dtterm` at the Prompt, and then press **Enter**, to open a Command Prompt.
2. Execute the following command: `<PRODUCT_HOME>/bin/bash wso2server.sh`. The operation log appears in the command window. When the product server has successfully started, the log displays the message "WSO2 Carbon started in 'n' seconds".

If you are starting the product in service/nohup mode in Solaris, do the following:

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

Once the server has started, you can run the Management Console by opening a Web browser and typing in the management console's URL. The URL is displayed as the last line in the start script's console and log. For example:

```
https://<Server Host>: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's installed, you can type "localhost" instead of the IP address: `https://localhost:9443/carbon`

Starting the server as a Worker Node:

If you want to provide access to the production environment without allowing any user group (including admin) to log in to the management console, you must start the product as a worker node. You can enable the worker node for your product in two ways:

1. Use the `-DworkerNode` startup command when you start the product as shown below:
   - **On Windows:** `<PRODUCT_HOME>/bin/wso2server.bat --run -DworkerNode`
   - **On Linux/Solaris:** `sh <PRODUCT_HOME>/bin/wso2server.sh -DworkerNode`

2. You can set permanently set the `-DworkerNode` system property to 'true' in your product startup script. When you execute the product startup script, the worker profile will be started automatically.
   ```
   '-DworkerNode=false'
   ```

If you want to check any additional options available to be used with the startup commands, type `-help` after the command, such as: `sh <PRODUCT_HOME>/bin/wso2server.sh -help`.

---

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

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

3. Start the product by following the above instructions.
At the sign-in screen, sign in to the Management Console using admin as both the username and password. You can then use the Management Console to manage the product. The tabs and menu items in the navigation pane on the left may vary depending on the features you have installed. For example, following is the Management Console for WSO2 Storage Server:

To view information about a particular page, click the Help link in the top right corner of that page, or click the Docs link to open this documentation for full information on managing the product.

When the Management Console Sign-in page appears, the web browser will typically display an "insecure connection" message, which requires your confirmation before you can continue.

The Management Console is based on 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, so 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 belongs to the given organization.

If you leave the Management Console unattended, the 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. Note that 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:

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

The `RemoteAddrValve` Tomcat valve defined in this file will only apply to the Carbon management console, and thereby all outside requests to the management console will be 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:

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

The `RemoteAddrValve` Tomcat valve defined in this file will apply to each web application hosted on the Carbon server. Therefore, all outside requests to any web application will be blocked.

You can also restrict access to particular servlets in a web application by adding a Remote Address Filter to the `web.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory), 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 page (`/carbon/admin/login.jsp`) is granted only to one IP address:

```xml
<filter>
  <filter-name>Remote Address Filter</filter-name>

  <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>

  <init-param>
    <param-name>allow</param-name>
    <param-value>127.0.01</param-value>
  </init-param>

</filter>

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

**Note:** Any configurations (including valves) defined in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file applies to all web applications and is globally available across server, regardless of host or cluster. See the official Tomcat documentation for more information about using remote host filters.
Stop the Server

To stop the server, press Ctrl+C in the command window, or click the Shutdown/Rstart link in the navigation pane in the Management Console.

Running the product on a Preferred Profile

When a WSO2 product server starts, it starts all features and related artifacts bundled in the product. Multi-profile support allows you to run the product on a selected profile so that only features specific to that profile along with common features start up with the server. This enables better resource utilization.

Execute one of the following commands to start a product on a preferred profile.

<table>
<thead>
<tr>
<th>OS</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td><code>&lt;PRODUCT_HOME&gt;/bin/wso2server.bat -Dprofile=&lt;preferred-profile&gt; --run</code></td>
</tr>
<tr>
<td>Linux/Solaris</td>
<td><code>sh &lt;PRODUCT_HOME&gt;/bin/wso2server.sh -Dprofile=&lt;preferred-profile&gt;</code></td>
</tr>
</tbody>
</table>

Given below are the profiles available in Data Services Server:

<table>
<thead>
<tr>
<th>DSS Profile</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>manager</td>
<td>Used when DSS acts as a manager node in a cluster. This profile starts front-end/UI features such as login as well as back-end services that allow the product instance to communicate with other nodes in the cluster.</td>
</tr>
<tr>
<td>worker</td>
<td>Used when DSS acts as a worker node in a cluster. This profile only starts the back-end features for data processing and communicating with the manager node.</td>
</tr>
</tbody>
</table>

How multi-profiling works

Starting a product on a preferred profile starts only a subset of features bundled in the product. In order to identify what feature bundles apply to which profile, each product maintains a set of bundles.info files in `<PRODUCT_HOME>/repository/components/<profile-name>/configuration/org.eclipse.equinox.simpleconfigurator` directories. The bundles.info files contain references to the actual bundles. Note that `<profile-name>` in the directory path refers to the name of the profile. For example, when there's a product profile named webapp, references to all the feature bundles required for webapp profile to function are in a bundles.info file saved in `<PRODUCT_HOME>/repository/components/webapp/configuration/org.eclipse.equinox.simpleconfigurator` directory.

Note that when you start the server without using a preferred profile, the server refers to `<PRODUCT_HOME>/repository/components/default/configuration/org.eclipse.equinox.simpleconfigurator/bundles.info` file by default. This file contains references to all bundles in `<PRODUCT_HOME>/repository/components/plugins` directory, which is where all components/bundles of a product are saved.

Product Administration

WSO2 Data Services Server (WSO2 DSS) is shipped with default configurations that will allow you to download,
install and get started with your product instantly. However, when you go into production, it is recommended to change some of the default settings to ensure that you have a robust system that is suitable for your operational needs. Also, you may have specific use cases that require specific configurations to the server.

If you are a product administrator, the follow content will provide an overview of the administration tasks that you need to perform when working with WSO2 Data Services Server (WSO2 DSS).

- Upgrading from a previous release
- Migrating DSS
- Changing the default database
- Configuring users, roles and permissions
- Configuring security
- Configuring transports
- Configuring multitenancy
- Configuring the registry
- Performance tuning
- Changing the default ports
- Installing, uninstalling and managing product features
- Configuring custom proxy paths
- Customizing error pages
- Configuring the management console
- Applying patches
- Error handling in DSS
- Timestamp conversion for RDBMS
- Enabling CORS for data services
- Monitoring the server

### Upgrading from a previous release

If you are upgrading from WSO2 DSS 3.5.0 to WSO2 DSS 3.5.1 version, see the upgrading instructions for DSS.

### Migrating DSS

If you want to migrate the data in your DSS from one environment to another, see Migrating DSS.

### Changing the default database

By default, WSO2 DSS is shipped with an embedded H2 database (Carbon database), which is used for storing user management and registry data. We recommend that you use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. when you set up your production environment. You can change the default database configuration by simply setting up a new physical database and updating the configurations in the product server to connect to that database.

- For information on setting up a new database for WSO2 DSS, see Setting up the Physical Database in the WSO2 Administration Guide.
- Once you set up a new database, you need to update the configurations in WSO2 DSS. See Changing the Carbon Database for instructions.

### Configuring users, roles and permissions

The user management feature in your product allows you to create new users and define the permissions granted to each user. You can also configure the user stores that are used for storing data related to user management.

- For instructions on how to configure user management, see Working with Users, Roles and Permissions in the WSO2 Product Administration Guide.
- For descriptions of permissions that apply to WSO2 DSS users, see Role-Based Permissions for WSO2 Data Services Server.

### Configuring security

After you install WSO2 DSS, it is recommended to change the default security settings according to the requirements of your production environment. As DSS 3.5.1 is built on top of the WSO2 Carbon Kernel (version 4.4.7), the main security configurations applicable to DSS are inherited from the Carbon kernel.
For instructions on configuring security in your server, see the following topics in the WSO2 Product Administration Guide.

- Configuring Transport-Level Security
- Using Asymmetric Encryption
- Using Symmetric Encryption
- Enabling Java Security Manager
- Securing Passwords in Configuration Files
- Resolving Hostname Verification

### Configuring transports

The transport management capability in WSO2 products allows a particular server to send and receive messages over several transport and application protocols. The following transports are supported in WSO2 DSS: **HTTP, HTTPS, JMS, and Local**.

For instructions on configuring these transport for WSO2 DSS, see the following topics in the WSO2 Product Administration Guide:

- HTTP Transport
- HTTPS Transport
- JMS Transport
- Local Transport

### Configuring multitenancy

You can create multiple tenants in your product server, which will allow you to maintain tenant isolation in a single server/cluster. For instructions on configuring multiple tenants for your server, see Working with Multiple Tenants in the WSO2 Product Administration Guide.

### Configuring the registry

A registry is a content store and a metadata repository for various artifacts such as services, WSDLs and configuration files. In WSO2 products, all configurations pertaining to modules, logging, security, data sources and other service groups are stored in the registry by default.

For instructions on setting up and configuring the registry for your server, see Working with the Registry in the WSO2 Product Administration Guide.

### Performance tuning

You can optimize the performance of your product server by configuring the appropriate OS settings, JVM settings etc. Most of these are server-level settings that will improve the performance of any WSO2 product. For instructions, see Performance Tuning in the WSO2 Product Administration Guide.

### Changing the default ports

When you run multiple WSO2 products, multiple instances of the same product, or multiple WSO2 product clusters on the same server or virtual machines (VMs), you must change their default ports with an offset value to avoid port conflicts.

For instructions on configuring posts, see Changing the Default Ports in the WSO2 Product Administration Guide.
Installing, uninstalling and managing product features

Each WSO2 product is a collection of reusable software units called features where a single feature is a list of components and/or other feature. By default, WSO2 DSS is shipped with the features that are required for your main use cases.

For information on installing new features, or removing/updating an existing feature, see Working with Features in the WSO2 Product Administration Guide.

Configuring custom proxy paths

This feature is particularly useful when multiple WSO2 products (fronted by a proxy server) are hosted under the same domain name. By adding a custom proxy path you can host all products under a single domain and assign proxy paths for each product separately.

For instructions on configuring custom proxy paths, see Adding a Custom Proxy Path in the WSO2 Product Administration Guide.

Customizing error pages

You can make sure that sensitive information about the server is not revealed in error messages, by customizing the error pages in your product. For instructions, see Customizing Error Pages in the WSO2 Product Administration Guide.

Customizing the management console

Some of the WSO2 products, such as WSO2 DSS consist of a web user interface named the management console. This allows administrators to configure, monitor, tune, and maintain the product using a simple interface. You can customize the look and feel of the management console for your product.

For instructions, see Customizing the Management Console in the WSO2 Product Administration Guide.

Applying patches

For instructions on applying patches (issued by WSO2), see WSO2 Patch Application Process in the WSO2 Product Administration Guide.

Error handling in DSS

See Error Handling in DSS for a list of the error messages that you may come across while working with WSO2 DSS.

Timestamp conversion for RDBMS

See Timestamp conversion for RDBMSs for instructions on configuring timestamp conversion.

Enabling CORS for data services

See Enabling CORS for data services for instructions.
Monitoring the server

Monitoring is an important part of maintaining a product server. Listed below are the monitoring capabilities that are available for WSO2 DSS.

- **Monitoring server logs**: A properly configured logging system is vital for identifying errors, security threats and usage patterns in your product server. For instructions on monitoring the server logs, see Monitoring Logs in the WSO2 Product Administration Guide.

- **JMX-based monitoring**: For information on monitoring your server using JMX, see JMX-based monitoring in the WSO2 Product Administration Guide.

Role-Based Permissions

This section explains in detail how the management console of a WSO2 product can be used for configuring the permissions granted to a user role. You will also find below, detailed descriptions on all the types of permissions that can be granted.

- Introduction to role-based permissions
- Configuring permissions for a role
- Descriptions of permissions
  - Log-in permissions
  - Super Tenant permissions
  - Tenant-level permissions
    - Permissions for configuring the server
    - Permissions for managing the server
    - Permissions for monitoring the server

Introduction to role-based permissions

The **User Management** module in WSO2 products enable role-based access. With this functionality, the permissions enabled for a particular role determines what that user can do using the management console of a WSO2 product. Permissions can be granted to a role at two levels:

- **Super tenant level**: A role with super tenant permissions is used for managing all the tenants in the system and also for managing the key features in the system, which are applicable to all the tenants.
- **Tenant level**: A role with tenant level permissions is only applicable to individual tenant spaces.

By default, every WSO2 product comes with the following User, Role and Permissions configured:

- The **Admin** user and **Admin** role is defined and linked to each other in the user-mgt.xml file, stored in the `<PRODUCT_HOME>/repository/conf/` directory as shown below.

```
<AddAdmin>true</AddAdmin>
<AdminRole>admin</AdminRole>
<AdminUser>
  <UserName>admin</UserName>
  <Password>admin</Password>
</AdminUser>
```

- The **Admin** role has all the permissions in the system enabled by default. Therefore, this is a super tenant, with all permissions enabled.

You will be able to log in to the management console of the product with the **Admin** user defined in the user-mgt.
You can then create new users and roles and configure permissions for the roles using the management console. However, note that you cannot modify the permissions of the Admin role. The possibility of managing users, roles and permissions is granted by the User Management permission.

Go to the WSO2 administration guide for detailed instructions on the following:

- Configuring the system administrator.
- Configuring users.
- Configuring roles.

Configuring permissions for a role

Once a user role is already created in your product, you can configure the permissions for the role as explained below:

1. Click Users and Roles in the Configure tab of the navigator. All the roles created in the system will be listed in the Roles page as shown below.

   Roles

<table>
<thead>
<tr>
<th>Name</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>Assign Users</td>
</tr>
<tr>
<td>Internal/everyone</td>
<td>Permissions</td>
</tr>
</tbody>
</table>

2. Click Permissions to open the permissions navigator for the role as shown below.

   Note that there may be other categories of permissions enabled for a WSO2 product, depending on the type of features that are installed in the product.

3. You can select the relevant check boxes to enable the required permissions for your role. The descriptions of all the available permissions are explained below.
Descriptions of permissions

Let us now go through each of the options available in the permissions navigator to understand how they apply to functions in WSO2 Data Services Server (WSO2 DSS).

- **Log-in permissions**
- **Super Tenant permissions**
- **Tenant-level permissions**
  - Permissions for configuring the server
  - Permissions for managing the server
  - Permissions for monitoring the server

**Log-in permissions**

The **Login** permission defined under **Admin** permissions allows users to log in to the management console of the product. Therefore, this is the primary permission required for using the management console.

**Super Tenant permissions**

The following table describes the permissions at **Super Tenant** level. These are also referred to as **Super Admin permissions**.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
</table>
| Configuration      | The **Super Admin/Configuration** permissions are used to grant permission to the key functions in a product server, which are common to all the tenants. In each WSO2 product, several configuration permissions will be available depending on the type of features that are installed in the product.  
  - **Feature Management** permission ensures that a user can control the features installed in the product using the management console. That is, the **Features** option will be enabled under the **Configure** menu. See the topic on **feature management** for more information.  
  - **Logging** permission enables the possibility to configure server logging from the management console. That is, the **Logging** option will be enabled under the **Configure** menu. See the topic on **configuring log4j properties** for more information. |
| Management         | The **Super Admin/Manage** permissions are used for adding new tenants and monitoring them.  
  - **Modify/Tenants** permission enables the **Add New Tenant** option in the **Configure** menu of the management console, which allows users to add new tenants.  
  - **Monitor/Tenants** permission enables the **View Tenants** option in the **Configure** menu of the management console.  
  See the topic on **configuring multiple tenants** for more information. |
| Server Admin       | Selecting the **Server Admin** permission enables the **Shutdown/Restart** option in the **Main** menu of the management console. |

**Tenant-level permissions**

The following table describes the permissions at **Tenant** level. These are also referred to as **Admin permissions**.

Note that when you select a node in the **Permissions** navigator, all the subordinate permissions that are listed under the selected node are also automatically enabled.
Permissions for configuring the server

The following table explains the permissions required for performing various configuration tasks in WSO2 DSS.

<table>
<thead>
<tr>
<th>Permission level</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin/Configure</td>
<td>When the Admin/Configure permission node is selected, the following menus are enabled in the management console:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Configure menu/Datasources</strong>: This permission enables the user to <strong>Add/Edit/Remove datasources</strong>.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Configure menu/Discovery</strong>: This permission enables the service discovery feature.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Configure menu/Theme</strong>: This permission is not applicable to WSO2 DSS.</td>
</tr>
<tr>
<td></td>
<td>- Additionally, all permissions listed under Configure in the permissions navigator are selected automatically.</td>
</tr>
<tr>
<td>Admin/Configure/Security</td>
<td>When the Admin/Configure/Security permission node is selected, the following menus are enabled in the Configure menu of the management console:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Keystores</strong>: See the topic on managing keystores for information.</td>
</tr>
<tr>
<td></td>
<td>- This permission will also enable the Roles option under Configure/Users and Roles. See the topic on configuring users, roles and permissions for more information.</td>
</tr>
<tr>
<td></td>
<td>- Additionally, all permissions listed under Security in the permissions navigator are selected automatically.</td>
</tr>
<tr>
<td>Admin/Configure/Security/Identity</td>
<td>This permission enables the possibility to add users from the Management Console. That is, the Users option will be enabled under Configure/Users and Roles. See the topic on configuring users, roles and permissions for more information.</td>
</tr>
<tr>
<td>Management/User Management</td>
<td></td>
</tr>
<tr>
<td>Admin/Configure/Security/Identity</td>
<td>This permission enables the profiles of all the users. You can view the profile in the Configure tab, Users and Roles -&gt; Users link.</td>
</tr>
<tr>
<td>Management/Profile Management</td>
<td></td>
</tr>
<tr>
<td>Admin/Configure/Security/Identity</td>
<td>This permission enables the Change Password option for the users listed in the User Management/Users and Roles/Users screen, which allows the user to change the passwords. See the topic on configuring users, roles and permissions for more information.</td>
</tr>
<tr>
<td>Management/Password Management</td>
<td></td>
</tr>
</tbody>
</table>

Permissions for managing the server

Listed below are the permissions for some of the general functions applicable to WSO2 DSS.

<table>
<thead>
<tr>
<th>Permission level</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Admin/Manage/Add

- **Manage menu/Add/Modules**: This permission enables you to upload modules using the management console.
- **Manage menu/Add/Services**: This permission enables you to upload/generate/create/schedule services in WSO2 DSS. See the tutorials on creating, generating, uploading data services and scheduling tasks.
- **Manage menu/Add/Webapps**: This permission is not applicable to WSO2 DSS.

## Add/Manage/Extensions

- **Manage menu/Extensions/List**: This permission is not applicable to WSO2 DSS.
- **Manage menu/Extensions/Add**: This permission is not applicable to WSO2 DSS.

## Add/Manage/Configure

- **Manage menu/Configure/Modules**: This permission enables listing of the modules.
- **Manage menu/Configure/Services**: This permission enables listing of the services.
- **Manage menu/Configure/Webapps**: This permission is not applicable to WSO2 DSS.

## Admin/Manage/Resources

This permission enables the **Browse** option under the **Registry** menu in the main navigator. This option allows users to browse the resources stored in the registry by using the **Registry** tree navigator.

See the topic on **working with the registry** for more information.

## Admin/Manage/Search

This permission enables the **Search** option under the **Registry** sub menu in the **Main** menu. This option allows users to search for specific resources stored in the registry by filling in the search criteria.

See the topic on **working with the registry** for more information.

### Permissions for monitoring the server

<table>
<thead>
<tr>
<th>Permission level</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Admin/Monitor</strong></td>
<td>When the <strong>Admin/Monitor</strong> permission node is selected, the following menus are enabled in the management console:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Monitor</strong> menu System Statistics: This allows users to monitor performance statistics.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Monitor</strong> menu/SOAP Tracer: This allows users to monitor SOAP messages.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Monitor</strong> menu/Message Flows: This allows users to monitor message flows.</td>
</tr>
<tr>
<td></td>
<td>- Additionally, all permissions listed under <strong>Monitor</strong> in the permissions navigator are selected automatically.</td>
</tr>
<tr>
<td><strong>Admin/Monitor/Logs</strong></td>
<td>When this node is selected, the following menus are enabled in the <strong>Monitor</strong> tab of the management console:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Monitor</strong> menu/System Logs: This allows users to monitor system logs.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Monitor</strong> menu/Application Logs: This allows users to application logs.</td>
</tr>
<tr>
<td></td>
<td>See the topic on <strong>viewing and downloading logs</strong> for instructions.</td>
</tr>
</tbody>
</table>

### Upgrading from a Previous Release

This page takes you through the steps for upgrading from DSS 3.5.0 to DSS 3.5.1. For more information on release
Preparing to upgrade

The following prerequisites must be completed before upgrading:

- Make a backup of the DSS 3.5.0 database and copy the `<DSS_HOME_3.5.0>` directory in order to backup the product configurations.

The downtime is limited to the time taken for switching databases when in the production environment.

Migrating the configurations

Since there are no database changes between these two DSS versions, you are only required to migrate the configurations and settings from DSS 3.5.0 to DSS 3.5.1 as explained below.

Updating the configuration files

The following are the updates that need to be done to the configuration files in DSS 3.5.1:

- Create a new database for DSS 3.5.1 and restore the backup of the old database in this new database.
- To connect DSS 3.5.0 to the new database, configure the following files:
  1. Configure the `<DSS_HOME_3.5.1>/repository/conf/datasources/master-datasources.xml` file as shown in the following example:
<datasource>
user manager</description>
<name>WSO2_CARBON_DB</name>
<description>The datasource used for registry and</description>
<jndiConfig>
<name>jdbc/WSO2CarbonDB</name>
</jndiConfig>
<definition type="RDBMS">
<configuration>
<url>jdbc:mysql://localhost:3306/<new_database></url>
$username username</username>
$password password</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>

2. Go to the $DSS_HOME_3.5.1$/repository/conf/ directory and update the datasource references
in the user-mgt.xml and registry.xml files to match the updated configurations in the master-datasources.xml file. The following are sample configurations if the datasource is "jdbc/WSO2CarbonDB":

registry.xml

$dbConfig name=wso2registry">
$dataSource>jdbc/WSO2CarbonDB</dataSource>
</dbConfig>

user-mgt.xml

<UserManager>
<Realm>
<Configuration>
... 
<Property
name=dataSource"jdbc/WSO2CarbonDB"></Property>
</Configuration>
... 
</Realm>
</UserManager>
3. Check for any other configurations that were done for DSS 3.5.0 (based on your solutions), and update the configuration files in DSS 3.5.1 accordingly. For example, external user stores, caching, mounting, etc.

Note that the properties of data services listed below have changed for all 3.x.x versions of WSO2 DSS. Therefore, if you have any .dbs files moved from an older DSS version (DSS 2.x.x version) to DSS 3.5.1, be sure to update these properties as applicable to the DSS 3.x.x versions. The following table shows how the properties have changed between DSS 2.x.x versions to DSS 3.x.x versions.

<table>
<thead>
<tr>
<th>DSS 2.x.x versions</th>
<th>DSS 3.x.x versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.wso2.ws.dataservice.driver</td>
<td>driverClassName</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.protocol</td>
<td>url</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.user</td>
<td>username</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.password</td>
<td>password</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.query_timeout</td>
<td>queryTimeout</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.autocommit</td>
<td>autoCommit</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.fetch_direction</td>
<td>fetchDirection</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.fetch_size</td>
<td>fetchSize</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.max_field_size</td>
<td>maxFieldSize</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.max_rows</td>
<td>maxRows</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.transaction_isolation</td>
<td>defaultTransactionIsolation</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.initial_size</td>
<td>initialSize</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.maxpoolsize</td>
<td>maxActive</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.max_idle</td>
<td>maxIdle</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.minpoolsize</td>
<td>minIdle</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.max_wait</td>
<td>maxWait</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.validation_query</td>
<td>validationQuery</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.test_on_borrow</td>
<td>testOnBorrow</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.test_on_return</td>
<td>testOnReturn</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.test_while_idle</td>
<td>testWhileIdle</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.time_between_eviction_runs_mills</td>
<td>timeBetweenEvictionRunsMillis</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.num_test_per_eviction_run</td>
<td>numTestsPerEvictionRun</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.min_evictable_idle_time_millis</td>
<td>minEvictableIdleTimeMillis</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.remove_abandoned</td>
<td>removeAbandoned</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.remove_abandoned_timeout</td>
<td>removeAbandonedTimeout</td>
</tr>
</tbody>
</table>
Migrating the tenant settings and applications

You can migrate all artifacts etc. relevant to tenants by copying the following directories from the old server to the new server.

1. Copy the following directories from the old database to the staging database.
   - To migrate the super tenant settings, copy the `<DSS_HOME>/repository/deployment/server` directory.
   - If multitenancy is used, copy the `<DSS_HOME>/repository/tenants/` directory.
2. Start the server.

Testing the upgrade

Verify that all the required scenarios are working as expected with DSS 3.5.1. This confirms that the upgrade is successful.

Migrating DSS

If you have multiple instances of DSS and you want to move your data and configurations from one instance to another (such as moving from development to test or production), you can create a Carbon Application project (C-App) and deploy the C-App in the new environment as described in Creating and Deploying a C-App. This approach works when the new instance you are migrating to uses the same DSS release and has the same database versions as the previous instance. Alternatively, you can copy all the data services configuration (.dbs) files from the deployment directory (`<PRODUCT_HOME>/repository/deployment/server/dataservices`) of the old environment to the new one.

See Upgrading from a Previous Release in the following situations:

- The new environment you are migrating to has a different database version. In this case, you must upgrade the older database.
- You want to upgrade from a previous DSS release to a new one.

Error Handling in DSS

WSO2 Data Services Server is used for exposing data (decoupled from the infrastructure) as a service, which can be used by external resources to conveniently access the data. When you use WSO2 DSS in an actual production environment, it is possible to encounter situations where the operations you invoke will result in a failed output. For example, you may have a wrong data setup or there may be erroneous operations invoked. In such instances, an error will be logged to indicate that the operation has failed.

Listed below are the error codes in WSO2 DSS and the situations in which they may occur.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.wso2.ws.dataservice.log_abandoned</td>
<td>logAbandoned</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.xa_datasource_class</td>
<td>dataSourceClassName</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.xa_datasource_properties</td>
<td>dataSourceProps</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.force_stored_proc</td>
<td>forceStoredProc</td>
</tr>
<tr>
<td>org.wso2.ws.dataservice.force_jdbc_batch_requests</td>
<td>forceJDBCBatchRequests</td>
</tr>
<tr>
<td>Exception</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DATABASE_ERROR</td>
<td>This exception is thrown at the point of invoking a request if there are errors occurring while querying the database or while processing the result of a query. For example, a syntax error thrown from the database, a primary key constraint violation, etc.</td>
</tr>
<tr>
<td>CONNECTION_UNAVAILABLE_ERROR</td>
<td>This error occurs at the point of creating a data service if the connection to the datasource cannot be established. You can find the instructions for adding datasources to your data service from here.</td>
</tr>
<tr>
<td>VALIDATION_ERROR</td>
<td>This error occurs when an input parameter in the request for a query fails the validation defined for that parameter. See the sample on validators to understand how validations work.</td>
</tr>
<tr>
<td>INCOMPATIBLE_PARAMETERS_ERROR</td>
<td>This error occurs in the following instances:</td>
</tr>
<tr>
<td></td>
<td>• When a request is invoked, the number of parameters passed in the request should match the number of input parameters defined in the query. Otherwise, there need to be default values specified in the data service (the .dbs file) for the parameters that are not passed in the request. If either of these requirements are not fulfilled, this error will occur at the point of invoking the request. See the sample on setting default parameter values to understand how this works.</td>
</tr>
<tr>
<td></td>
<td>• When a request is invoked, the list of 'with-params' given for the operation defined in your data service should match the input parameters given in the corresponding query definition. If there is a mismatch, this error will occur. Read more about how these parameters are used when defining operations and queries in a data service.</td>
</tr>
<tr>
<td>UNKNOWN_ERROR</td>
<td>For all other types of errors.</td>
</tr>
</tbody>
</table>

**Performance Tuning**

This section describes some recommended performance tuning configurations to optimize WSO2 DSS. It assumes that you have set up WSO2 DSS on a server running Unix/Linux, which is recommended for a production deployment.

- OS-level settings
- JVM settings
- JDBC pool configuration
- Setting the thread execution limit for multitenant mode
- Other tips

**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 DSS accordingly.
OS-level settings

1. To optimize network and OS performance, configure the following settings in `/etc/sysctl.conf` file of Linux. These settings specify a larger port range, a more effective TCP connection timeout value, and a number of other important parameters at the OS-level.

```
net.ipv4.tcp_fin_timeout = 30 fs.file-max = 2097152
net.ipv4.tcp_tw_recycle = 1 net.ipv4.tcp_tw_reuse = 1
net.core.rmem_default = 524288 net.core.wmem_default = 524288
net.core.rmem_max = 67108864 net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 87380 16777216 net.ipv4.tcp_wmem = 4096 65536
16777216 net.ipv4.ip_local_port_range = 1024 65535
```

When we have the localhost port range configuration lower bound to 1024, there is a possibility that some processes may pick the ports which are already used by WSO2 servers. Therefore, it's good to increase the lower bound as sufficient for production, e.g., 10,000.

2. To alter the number of allowed open files for system users, configure the following settings in `/etc/security/limits.conf` file of Linux.

```
* soft nofile 4096 * hard nofile 65535
```

Optimal values for these parameters depend on the environment.

JVM settings

If one or more worker nodes in a clustered deployment require access to the management console, you need to increase the entity expansion limit in the `<DSS_HOME>/bin/wso2server.bat` file (for windows) or the `<DSS_HOME>/bin/wso2server.sh` file (for Linux/Solaris) as show below. The default entity expansion limit is 64000.

```
-DentityExpansionLimit=100000
```

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 goal of tuning the pool properties is to maintain a pool that is large enough to handle peak load without unnecessarily utilising resources. These pooling configurations can be tuned for your production server in general in the `<DSS_HOME>/repository/conf/datasources/master-datasources.xml` file. You can separately tune the configurations for the RDBMS datasources at the time of creating the datasource using the management console. Read about configuring an RDBMS datasource and creating a data service with RDBMS datasource for details.

The following parameters should be considered when tuning the connection pool:

- The application's concurrency requirement.
- The average time taken to run a database query.
- The maximum number of connections the database server can support.

The table below indicates some recommendations on how to configure the JDBC pool. For more details about
recommended JDBC configurations, see [Tomcat JDBC Connection Pool](#).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Tuning Recommendations</th>
</tr>
</thead>
</table>
| `maxActive` | The maximum number of active connections that can be allocated from the connection pool at the same time. The default value is 100. | The maximum latency (approximately) = \((P / M) * T\),  
where,  
- \(M\) = maxActive value  
- \(P\) = Peak concurrency value  
- \(T\) = Time (average) taken to process a query.  
Therefore, by increasing the maxActive value (up to the expected highest number of concurrency), the time that requests wait in the queue for a connection to be released will decrease. But before increasing the Max. Active value, consult the database administrator, as it will create up to maxActive connections from a single node during peak times, and it may not be possible for the DBMS to handle the accumulated count of these active connections.  
Note that this value should not exceed the maximum number of requests allowed for your database. |
| `maxWait` | The maximum time that requests are expected to wait in the queue for a connection to be released. This property comes into effect when the maximum number of active connections allowed in the connection pool (see maxActive property) is used up. | Adjust this to a value slightly higher than the maximum latency for a request, so that a buffer time is added to the maximum latency. That is,  
If the maximum latency (approximately) = \((P / M) * T\),  
where,  
- \(M\) = maxActive value,  
- \(P\) = Peak concurrency value,  
- \(T\) = Time (average) taken to process a query,  
then, the maxWait = \((P / M) * T + \) buffer time. |
| `minIdle` | 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. |
| `maxIdle` | The maximum number of connections that can remain idle in the pool. | The value should be less than the maxActive value. For high performance, tune maxIdle to match the number of average, concurrent requests to the pool. If this value is set to a large value, the pool will contain unnecessary idle connections. |
testOnBorrow | The indication of whether connection objects will be validated before they are borrowed from the pool. If the object validation fails, the connection is dropped from the pool, and there will be an attempt to borrow another connection.

When the connection to the database is broken, the connection pool does not know that the connection has been lost. As a result, the connection pool will continue to distribute connections to the application until the application actually tries to use the connection. To resolve this problem, set “Test On Borrow” to “true” and make sure that the “ValidationQuery” property is set. To increase the efficiency of connection validation and to improve performance, validationInterval property should also be used.

validationInterval | This parameter controls how frequently a given validation query is executed (time in milliseconds). The default value is 30000 (30 seconds). That is, if a connection is due for validation, but has been validated previously within this interval, it will not be validated again.

Deciding the value for the “validationInterval” depends on the target application’s behavior. Therefore, selecting a value for this property is a trade-off and ultimately depends on what is acceptable for the application.

If a larger value is set, the frequency of executing the Validation Query is low, which results in better performance. Note that this value 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 the validation query execution is usually fast. Therefore, even if this value is only large by a few seconds, there will not be a big penalty on performance. Also, specially when the database requests have a high throughput, the negative impact on performance is negligible. For example, a single extra validation query run every 30 seconds is usually negligible.

If a smaller value is set, a stale connection will be identified quickly when it is presented. This maybe important if you need connections repaired instantly, e.g. during a database server restart.

validationQuery | The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw an SQLException. The default value is null. Example values are SELECT 1(mysql), select 1 from dual(oracle), SELECT 1(MS Sql Server).

Specify an SQL query, which will validate the availability of a connection in the pool. This query is necessary when testOnBorrow property is true.

When it comes to web applications, users are free to experiment and package their own pooling framework such BoneCP.

Setting the thread execution limit for multitenant mode

In multitenant mode, the 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 `<DSS_HOME>/repository/conf/tomcat/catalina-server.xml` file as shown below.

```
<Vale
   className="org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValuve" threshold="600"/>
```

- The `className` is the Java class name used for the implementation. This must be set to `org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve`.
- The `threshold` gives the minimum duration in seconds after which a thread is considered stuck. Default value is 600 seconds.

Other tips

When you need high update/insert throughput, use batch requests to achieve maximum performance. Batch requests send multiple records in a single request. For more information on batch processing, see [Batch Processing Sample](#).

**Timestamp Conversion for RDBMS**

By default, DSS converts the timestamp to UTC time zone before inserting any timestamp data to the database via DSS. When retrieving the data, DSS will convert the timestamp back to the server timezone. This can sometimes lead to inconsistencies where the time zones of the timestamps for inserting data and retrieving data are different. Therefore, you can disable UTC conversion for RDBMSs as follows.

1. Open the server startup script, which is stored in the `<DSS_HOME>/bin` directory.
   - For Linux: `wso2server.sh`
   - For Windows: `wso2server.bat`
2. Set the following property to false: `-Ddss.legacy.timezone.mode=true`

This configuration ensures that data is entered into RDBMSs via DSS using the server timezone instead of converting to UTC.

**Enabling CORS for Data Services**

You can enable Cross Origin Resource Sharing for data services deployed in WSO2 DSS. As explained below, you have the option of enabling CORS for selected data services or for all the data services.

To enable CORS in DSS:

1. Download the following JARs: "cors-filter-2.4.jar" and "java-property-utils-1.9.1.jar".
2. Copy the JARs to the `<DSS_HOME>/repository/components/lib/` directory.
3. Add the following configurations to the `<DSS_HOME>/repository/conf/tomcat/web.xml` file.
4. Edit the `<filter-mapping>` section in the above configuration to specify whether CORS should be enabled for all data services or only selected data services.

- To enable CORS for a selected data service, add the service's url as the url pattern:

```
<filter-mapping>
    <filter-name>CORS</filter-name>
    <url-pattern>/services/example/*</url-pattern>
</filter-mapping>
```

- To enable CORS for multiple data services, you add the urls in a comma separated list:

```
<filter-mapping>
    <filter-name>CORS</filter-name>
    <url-pattern>/services/sampleservice1/*,/services/sampleservice2/*</url-pattern>
</filter-mapping>
```

Alternatively, you can add two separate filter mappings:

```
<filter-mapping>
    <filter-name>CORS</filter-name>
    <url-pattern>/services/sampleservice1/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>CORS</filter-name>
    <url-pattern>/services/sampleservice2/*</url-pattern>
</filter-mapping>
```

5. Restart the server.

**Java Documentation**

The Java document Data Services Core describes all the classes, interfaces, and methods of the Data Services Core API, which you can use to create custom classes for tasks, user authenticators, and more.
Working with Data Services

If you are new to WSO2 Data Services Server (WSO2 DSS), go to the Tutorials section and run through the main tutorials in order to understand how the product works. The following topics will explain some of the advanced options that you can use when working with data services in WSO2 DSS.

- Defining Namespaces
- Managing Data Services
- Security Implementation
- Using JSON Messages with RESTful Services
- Working with Message Builders and Formatters
- Working with RDBMS
- Writing Advanced Data Service Queries

Defining Namespaces

Data services are implemented using XML, which means that there can be conflicting element names. Therefore, we use namespaces to uniquely identify the elements defined within a specific web service. Using namespaces for a data service is optional. However, in certain data services, namespaces become necessary. Read below for more information.

Namespace inheritance in a data service:

Namespaces can be defined at different levels of a data service, which means that you can have multiple namespaces applicable to your data service at the same time. In such a scenario, the top-level namespaces will be inherited by the lower level as explained below.

1. The namespace at the data service level applies to all aspects of the data service, i.e., to all elements in the results of all the queries (output mapping).
2. The namespace at the query result level (row namespace) overrides the service namespace and applies a given namespace to all the result elements (output mapping elements) of that query.
3. The namespace specified for a result element (output mapping) overrides the query-level row namespace.
4. If you have a complex element with child elements nested:
   1. The namespace for the complex element overrides the result row namespace and assigns the namespace to all its child elements.
   2. The namespace specifically assigned to a child element overrides the parent element's namespace.

See the following topics for instructions on setting namespaces at different levels of a data service:

- Setting the namespace for a data service
- Setting the namespace for a data service query
- Setting namespaces for specific elements in the query result
- Setting namespaces for complex results

Setting the namespace for a data service

The service namespace is used to uniquely identify a web service, which means that this namespace applies to all elements in the data service. In the WSDL that represents the service, the namespace is indicated by the "targetNamespace" attribute. The service namespace can be set when you define a data service using the management console as shown below.
Setting the namespace for a data service query

When you define a query for your data service, you can set a namespace for the elements in the query result (output mapping). That is, the **Row Namespace** applies to all the result rows that you define as output mappings for that query. The example below shows how the row namespace is set for the query result element: **Customers**.

After setting the row namespace, the data type of the **Customers** element falls under the given namespace as shown below.
If you have multiple queries with the same element wrapper but different element rows, be sure to use separate namespaces for each query. The example below illustrates two queries in a data service (.dbs file) with the following qualities:

- The element wrapper is the same for both queries, which is `<result element="employees" rowName="employee">`.
- The two queries contain two unique element rows: 'salary' and 'phone'.

Therefore, since we are using the same element wrapper for both queries, we need to use two separate namespaces to identify the two conflicting elements ('salary' and 'phone') in the two queries as shown below.
Setting namespaces for specific elements in the query result

The row namespace applies to all elements in the query result. Alternatively, you can specify namespaces for selected elements (output mapping elements) in the query result as shown below. This namespace will only apply to that element.
This is how the setting is reflected in the WSDL.

```xml
<wSDL:complexType name="customer">  
  <wss:sequence>  
    <wss:element maxOccurs="unbounded" minOccurs="0" name="customer" type="ns1:customer" />  
  </wss:sequence>  
</wSDL:complexType>  
<wSDL:complexType name="contact">  
  <wss:sequence>  
    <wss:element name="contact-last-name" nillable="true" type="wss:string" />  
    <wss:element name="contact-first-name" nillable="true" type="wss:string" />  
    <wss:element name="phone" nillable="true" type="wss:string" />  
    <wss:element name="city" nillable="true" type="wss:string" />  
    <wss:element name="country" nillable="true" type="wss:string" />  
  </wss:sequence>  
</wSDL:complexType>  
<wSDL:complexType name="complexType">  
  <wss:sequence>  
    <wss:element name="contact-name" nillable="true" type="wss:string" />  
  </wss:sequence>  
</wSDL:complexType>  
</wSDL:definition>
</wSDL:definitions>
```

If you have a data service query using the same result element multiple times, be sure to use separate namespaces for each element to avoid conflicts. The example below illustrates a single query in a data service (.dbs file) that has the same element defined twice. In this situation, we have to use separate namespaces for the two elements in order to distinguish them separately.

```xml
<query id="getProductByIDSQL" useConfig="default">  
  <sql>select * from Products where productID = ?</sql>  
  <result defaultNamespace="http://abc.com/productIDNamespace" element="products" rowName="product">  
    <element column="productID" name="productID" xsdType="string"/>  
    <element column="productID" name="productID" xsdType="INTEGER"/>  
    <element column="productName" name="productName" xsdType="string"/>  
  </result>
</query>
```

### Setting namespaces for complex results

In complex results, the result elements can be arbitrarily nested. These nested elements can have their own namespaces. The example given below illustrates a query with a complex result (nested elements). As shown below, the namespace specified for the 'contact' element is inherited by the child elements that are nested within. However, if you assign specific namespaces to the child elements, the namespace inherited by the parent element will be overridden.
<query id="customersInBostonSQL" useConfig="default">
  <sql>
    select * from Customers where city = 'Boston' and country = 'USA'
  </sql>
  <result defaultNamespace="http://abc.com/nsb" element="customers"
    rowName="customer">
    <element column="customerName" name="customer-name" xsdType="string"/>
    <element namespace="http://abc.com/nsb" name="contact">
      <element column="contactLastName" name="contact-last-name" xsdType="string"/>
      <element column="contactFirstName" name="contact-first-name" xsdType="string"/>
    </element>
    <element column="phone" name="phone" xsdType="string"/>
    <element column="city" name="city" xsdType="string"/>
    <element column="country" name="country" xsdType="string"/>
  </result>
</query>

The WSDL created for the above service is as follows:

The resulted XML of the executed service is as follows:
Managing Data Services

After a data service is created, you can manage it using its dashboard operations. The following topics explain data service management:

- Service Dashboard
- Service Group Dashboard
- Filtering Content to Specific User Roles
- Editing a Data Service

Service Dashboard

After creating a data service, you can configure its quality of service (QoS) operations and manage and monitor it using the dashboard. Follow the steps below to access the dashboard of a service.

1. Log in to the management console and select Services > List under the Main menu.
2. The Deployed Services page appears, which lists out all deployed services. Click on a service to view its dashboard.

A typical deployed service’s dashboard provides the features shown below. Some of these features may not be available for WIP services.

- Service details
- Client operations
- Statistics
Service details

The **Service Details** panel provides the following information:

- **Service Name** - Unique name of the service
- **Service Description** - Briefly describes the service
- **Service Group Name** - The name of the group the service belongs to. See Service Group Dashboard
- **Deployment Scope**
- **Service Type** Specifies service type (for example, Axis2, Proxy, STS)

Client operations

A service's dashboard provides the following client operations:

- **Try this service**
- **Generate client**
- **WSDL 1.1** and **WSDL 2.0**

It also lists the endpoints of a service. The service is set to two endpoints by default. Typically, the endpoint URI is of the format `http://{host ip of the server}:{server port}/services/{service name}`

**Try this service**

The **Try this service** link directs you to a new window to test if the service gives the desired output. The URL for invoking the **TryIt** tool can also be appended by concatenating the endpoint URL of your service with the following string:

- ?tryit : Ajax client for simple, generic interactions with the service
For more information, see Testing WSDLs.

Generate client

The Generate Client option is provided by the wsdl.tools feature of the WSO2 feature repository. For more information, see Generate Java Code for WSDL.

WSDL 1.1 and WSDL 2.0

Opens XML configurations. The WSDLs describe the operations the service exposes, the structure of the XML that is sent and received by each operation, and how to communicate with the service to retrieve the XML content. By default, six different types of endpoints are deployed. Each of these endpoint types are represented by the WSDL 2.0 <endpoint> elements of the service, and by the <binding> elements that these endpoints refer to. The default six types of endpoints are as follows:

- SOAP 1.2 over HTTP
- SOAP 1.1 over HTTP
- Plain old XML/REST over HTTP
- SOAP 1.2 over HTTPS
- SOAP 1.1 over HTTPS
- Plain old XML/REST over HTTPS

The URLs for the WSDL files of the service can also be appended by concatenating the endpoint URL of the service with the following strings:

- ?wsdl2 : WSDL 2.0 description of the service
- ?wsdl : WSDL 1.1 description of the service

Statistics

The Statistics panel provides statistical information about a service as follows:

- Request Count
- Response Count
- Fault Count
- Maximum Response Time
- Minimum Response Time
- Average Response Time

It also provides a graphical view of the system response time. For more information, see Access and Performance Statistics.

Service Group Dashboard

A Service Group is a convenient way of deploying multiple services in one service archive file. There is a logical relationship between the services at runtime. The only difference in the services.xml for a service group and a single service is its root element. For a service group, the root element is <serviceGroup>, and we have multiple <service> elements inside the <serviceGroup> element. For example,
The steps below show how to access and manage service groups.

1. Log in to the management console and select Services > List under the Main menu.
2. In the Deployed Services page that appears, click the deployed service group(s) link to access the groups.

3. The Deployed Service Groups window opens. Click a group to go to its dashboard.
4. A typical service group dashboard contains the following services:

**Services**

The Services Panel lists all the services included in the group. For example,

```
<serviceGroup>
  <service name=Test1>...
  </service>
  <service name=Test2>...
  </service>
</serviceGroup>
```

From here you can:

- Click a service's name and view its dashboard
- View the service's WSDLs
- Test the service using the try this service link

**Filtering Content to Specific User Roles**

Using the Data Services Server, you can control access to sensitive data for specific user roles. This facility is called Role-based content filtering. It filters data where specific data sections are only accessible by a given type of users.

Follow the steps below to filter a data service according to a specific user role.

1. Log on to the management console and select Services > List under the Main menu.
2. The Deployed Services window appears, which lists out all the currently active services and service groups. Click the data service you want to edit to open its dashboard.
3. In the dashboard, click the Edit Data Service link.

You can directly edit the XML file or use the management console. We use the latter in this example.

4. Navigate to the Queries page of the data service, select the query you want to edit and Edit Query.
5. In the **Output Mappings** section of the query, edit the field that needs to be filtered and tick the appropriate user role in the **Allowed User Roles** section.

6. Once all the required roles are set, the result entries of the **Edit Query** page look as follows in this example:
Editing a Data Service

After creating and deploying a service, you can change it as follows:

1. Log in to the management console and select Services > List under the Main menu.
2. The Deployed Services window appears, which lists out all the currently active services and service groups. Click the data service you want to edit to open its dashboard.
3. In the dashboard, click the Edit Data Service link. You can directly edit the XML file of the service using the information here or use the management console. The management console walks you through the same steps discussed here.

Every time you edit a service, you must redefine its security policies. This is because the security policies get removed when the service gets changed and re-deployed in the server.

Security Implementation

Web services security, or to be more precise, SOAP message security identifies and provides solutions for general computer security threats as well as threats unique to Web services. WSO2 supports WS Security, WS-Policy and WS-Security Policy specifications. These specifications define a behavioral model for Web services. Since a requirement for one Web service may not be valid for another, the Data Services Server also helps define service-specific security.

It provides 16 predefined, commonly-used security scenarios. All you have to do is apply the required security scenario into your service through the service's dashboard. You can also define a custom security policy. Understanding the exact security requirements is the first step in planning to secure Web services. Consider what security aspects are important to your service, whether it is the integrity, confidentiality, or both.
The default security scenarios

The topics below explain the 16 default security scenarios provided by WSO2.

1. UsernameToken
2. Non-repudiation
3. Integrity
4. Confidentiality
5. Sign and encrypt - X509 Authentication
6. Sign and Encrypt - Anonymous clients
7. Encrypt only - Username Token Authentication
8. Sign and Encrypt - Username Token Authentication
9. SecureConversation - Sign only - Service as STS - Bootstrap policy - Sign and Encrypt , X509 Authentication
10. SecureConversation - Sign Only - Service as STS - Bootstrap policy - Sign and Encrypt , Anonymous clients
11. SecureConversation - Sign and Encrypt - Service as STS - Bootstrap policy - Sign and Encrypt , X509 Authentication
12. SecureConversation - Sign Only - Service as STS - Bootstrap policy - Sign and Encrypt , Anonymous clients
13. SecureConversation - Sign and Encrypt - Service as STS - Bootstrap policy - Sign and Encrypt , Anonymous clients
14. SecureConversation - Encrypt Only - Service as STS - Bootstrap policy - Sign and Encrypt , Username Token Authentication
15. SecureConversation - Sign and Encrypt - Service as STS - Bootstrap policy - Sign and Encrypt , Username Token Authentication
16. Kerberos Token-based Security

1. UsernameToken

![Diagram of Username Token over HTTPS]

**Username Token over HTTPS**
Message is secured at the Transport level; Username Token is used for authentication. Client sends the Username Token inside the message which is validated against the entries in user store of the service’s end.

2. Non-repudiation
3. Integrity

**Non-repudiation Scenario**
Clients should have X.509 certificates and Messages are signed using the private key of the sender and verified using the public key of the signing party.
4. Confidentiality

Integrity Scenario
Clients do not have X.509 certificates; message is signed using a symmetric key derived by Client, which is encrypted using the service’s public key and sent as an encrypted key to the service inside the SOAP message. Service can decrypt the encrypted key using its private key and verify the signature using the symmetric key.
Confidentiality Scenario
Clients do not have X.509 certificates; message is encrypted using a symmetric key derived by Client, which is encrypted using the service's public key and sent as an encrypted key to the service inside the SOAP message. Service can decrypt the encrypted key using its private key and decrypt the message using the symmetric key.

5. Sign and encrypt - X509 Authentication
"Sign and Encrypt – X.509 Authentication” Scenario

Client’s should have X.509 certificates; messages are encrypted using recipient’s public key and signed using sender’s private key. At the recipient’s end, messages are decrypted using recipient’s private key and message signatures are validated using sender’s public key. Each party should have the public certificate of the other party in its key store.

Certificate alias and private key password of the client’s certificate are used to authenticate the client. Clients should provide that information in order to sign the message.

6. Sign and Encrypt - Anonymous clients
"Sign and Encrypt – Anonymous Clients” Scenario
Cients do not have X.509 certificates; message is signed and encrypted using a symmetric key derived by Client, which is encrypted using the service's public key and sent as an encrypted key to the service inside the SOAP message. Service can decrypt the encrypted key using its private key and get the symmetric key. Then service can use that symmetric key to decrypt the message and validate the signature.

7. Encrypt only - Username Token Authentication
8. Sign and Encrypt - Username Token Authentication

"Encrypt Only - Username Token Authentication" Scenario
Clients do not have X.509 certificates, message is encrypted using a symmetric key derived by Client, which is encrypted using the service's public key and sent as an encrypted key to the service inside the SOAP message. Service can decrypt the encrypted key using its private key and get the symmetric key. Then service can use that symmetric key to decrypt the message.

Authentication is achieved by using Username Tokens. Clients send the Username Tokens inside a message and Service validates them against the entries in its user store.
9. SecureConversation - Sign only - Service as STS - Bootstrap policy - Sign and Encrypt, X509 Authentication

“Sign and Encrypt - Username Token Authentication” Scenario

Clients do not have X.509 certificates; message is signed and encrypted using a symmetric key derived by Client, which is encrypted using the service’s public key and sent as an encrypted key to the service inside the SOAP message. Service can decrypt the encrypted key using its private key and get the symmetric key. Then service can use that symmetric key to decrypt the message and validate the signature.

Authentication is achieved by using UsernameTokens. Clients send the Username Tokens inside a message and Service validates them against the entries in its user store.
"Secure Conversation – Sign Only – Service as STS – Sign and Encrypt, X509 Authentication"

Clients should have X.509 certificates; they are establishing a security context with the service by using this certificate. RST and RSTR are signed and encrypted using PKI of service and client.

Once the security context is established, keys derived from the security context are used to sign the messages between service and client.

10. SecureConversation - Sign Only - Service as STS - Bootstrap policy - Sign and Encrypt, Anonymous clients
"Secure Conversation – Encrypt Only – Service as STS – Sign and Encrypt, X509 Authentication”

Clients should have X.509 certificates; they are establishing a security context with the service by using this certificate. RST and RSTR are signed and encrypted using PK1 of service and client.

Once the security context is established, keys derived from the security context are used to encrypt the messages between service and client.

11. SecureConversation - Sign and Encrypt - Service as STS - Bootstrap policy - Sign and Encrypt, X509 Authentication
"Secure Conversation – Sign and Encrypt – Service as STS – Sign and Encrypt, X509 Authentication"

Clients should have X.509 certificates; they are establishing a security context with the service by using this certificate. RST and RSTR are signed and encrypted using PKI of service and client.

Once the security context is established, keys derived from the security context are used to sign and encrypt the messages between service and client.

12. SecureConversation - Sign Only - Service as STS - Bootstrap policy - Sign and Encrypt , Anonymous clients
13. SecureConversation - Sign and Encrypt - Service as STS - Bootstrap policy - Sign and Encrypt, Anonymous clients

"Secure Conversation – Sign Only – Service as STS – Sign and Encrypt, Anonymous Clients"

Clients do not have X.509 certificates; they are establishing a security context with the service by using a symmetric binding.

Once the security context is established, keys derived from the security context are used to sign the messages between service and client.
"Secure Conversation – Encrypt Only – Service as STS – Sign and Encrypt, Anonymous Clients"

Clients do not have X.509 certificates; they are establishing security context with the service by using a symmetric binding.

Once the security context is established, keys derived from the security context are used to encrypt the messages between service and client.

14. SecureConversation - Encrypt Only - Service as STS - Bootstrap policy - Sign and Encrypt , Username Token Authentication
"Secure Conversation – Encrypt Only – Service as STS – Sign and Encrypt, Username Token Authentication”

Clients do not have X.509 certificates; they are establishing a security context with the service by using a symmetric binding. Username Token is used to authenticate the user when establishing the security context.

Once the security context is established, keys derived from the security context are used to encrypt the messages.

15. SecureConversation - Sign and Encrypt - Service as STS - Bootstrap policy - Sign and Encrypt, Username Token Authentication
16. Kerberos Token-based Security

If you apply security scenario 16 (Kerberos Token-based Security), you must associate your service with a service principal. Security scenario 16 is only applicable if you have a Key Distribution Center (KDC) and an Authentication Server in your environment. Ideally you can find KDC and an Authentication Server in a LDAP Directory server.

Two configuration files are used to specify Kerberos related parameters as follows.

- **krb5.conf** - Includes KDC server details, encryption/decryption algorithms etc.
- **jaas.conf** - Includes information relevant to authorization.

The above files are located in `<PRODUCT_HOME>/repository/conf/security` folder.

After selecting scenario 16, fill information about the service principal to associate the Web service with. You must specify the service principal name and password. The service principal must be already defined in the LDAP Directory server.

**Using JSON Messages with RESTful Services**
The tutorial on exposing data as a REST resource guides you with step-by-step instructions on how to expose data as REST-style resources. By default, the REST resources or data services defined in WSO2 DSS receives messages and responds to messages in XML format. However, you have the option of specifying the response format when you define your data service/resource. See the topic on output mapping for instructions.

Given below are some guidelines that you need to follow when working with REST resources and JSON messages in WSO2 DSS.

- Receiving a JSON response from a REST resource
- Sending JSON payloads to a REST resource

Receiving a JSON response from a REST resource

From WSO2 DSS 3.2.0 onwards, the possibility of receiving messages in JSON format is enabled by default. Therefore, you can simply send requests by adding “Accept: Application/json” to the request header, and you will receive the response in JSON. For example, shown below is how you can send a GET request to a resource using CURL:

```
```

Additionally, if you have the output mapping set to JSON in your resource query as explained here, the JSON response you receive will be according to the JSON template you have defined in the output mapping.

The possibility of receiving JSON responses is enabled by the following settings in the axis2.xml and axis2_client.xml files (stored in the <DSS_HOME>/repository/conf/axis2 directory):

```
<parameter name="httpContentNegotiation">true</parameter>
```

Sending JSON payloads to a REST resource

If you are sending JSON request payloads for data services REST resources, the JSON payload must be in a specific format for different types of HTTP methods as explained below.

**HTTP POST**

When you send an HTTP POST request, the format of the JSON object name should be "_post$RESOURCE_NAME", and the child name/values of the child fields should be the names and values of the input parameters in the target query. Given below is an example of a JSON request that sends an HTTP POST request to the "employee" resource path.
HTTP PUT

When you send an HTTP PUT request, the format of the JSON object name should be "_put$RESOURCE_NAME", and the child name/values of the child fields should be the names and values of the input parameters in the target query. Given below is an example of a JSON request that sends an HTTP PUT request to the "employee" resource path.

```
{
  "_putemployee": {
    "employeeNumber" : 1002,
    "lastName": "Smith",
    "firstName": "Will",
    "email": "will@smith.com",
    "salary": 86500.0
  }
}
```

JSON-based resource batch requests

When batch requests are enabled for data services resources, resource paths are created with the "_batch_req" suffix. In the payload content, the single request JSON object becomes one of the many possible objects in a parent JSON array object. This will have a name with a single request wrapper object + the "_batch_req". Shown below is a sample batch request for the HTTP POST request shown earlier.

```
{
  "_batch_req": {
    "_postemployee": {
      "employeeNumber" : 1,
      "lastName": "Doe",
      "firstName": "John",
      "email": "jdoe@wso2.com",
      "salary": 13500.0
    }
  }
}
```
Working with Message Builders and Formatters

When a message is received by WSO2 DSS, the receiving transport selects a message builder based on the content type of the message. The selected builder will then be used to process the message's raw payload data and convert it into SOAP. For example, if a JSON type message is received (contentType set to 'application/json'), the message builder corresponding to JSON will convert the message into SOAP.

Conversely, when a message is sent from WSO2 DSS in response to an HTTP request, a message formatter corresponding to the accept header of the HTTP request is used to convert the outgoing message into the requested data format. For example, when a JSON type message is requested by an HTTP request (accept header set to 'application/json'), the message formatter corresponding to JSON will be used to build the outgoing message in JSON.

Message builders and message formatters are thereby used to support content negotiation.

- Configuring message builders and formatters
  - Default message builders
  - Default message formatters

Configuring message builders and formatters

Message builders and formatters are configured in the axis2.xml file stored in the <DSS_HOME>/repository/conf/axis2 folder under the <messageBuilder> and <messageFormatters> sections. Given below are the default message builders and message formatters that are configured in the axis2.xml file.

Default message builders

Following are the default message builders configured in WSO2 DSS and the corresponding content types of incoming messages:
<table>
<thead>
<tr>
<th>Content type</th>
<th>Example</th>
<th>Message Builder</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/x-www-form-urlencoded</td>
<td>form</td>
<td>org.apache.axis2.builder.XFormURLEncodedBuilder</td>
</tr>
<tr>
<td>multipart/form-data</td>
<td>form</td>
<td>org.apache.axis2.builder.MultipartFormDataBuilder</td>
</tr>
<tr>
<td>application/xml</td>
<td>Plain old xml</td>
<td>org.apache.axis2.builder.ApplicationXMLBuilder</td>
</tr>
<tr>
<td>application/json</td>
<td>JSON (mapped notation)</td>
<td>org.apache.axis2.json.gson.JsonBuilder</td>
</tr>
<tr>
<td>application/json/badgerfish</td>
<td>JSON (badgerfish notation)</td>
<td>org.apache.axis2.json.JSONBadgerfishOMBuilder</td>
</tr>
<tr>
<td>text/javascript</td>
<td>Json payload</td>
<td>org.apache.axis2.json.gson.JsonBuilder</td>
</tr>
</tbody>
</table>

**Default message formatters**

Following are the default message formatters configured in WSO2 DSS and the corresponding accept headers used for outgoing messages.

<table>
<thead>
<tr>
<th>Accept type</th>
<th>Example</th>
<th>Message Formatters</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/x-www-form-urlencoded</td>
<td>forms</td>
<td>org.apache.axis2.transport.http.XFormURLEncodedFormatter</td>
</tr>
<tr>
<td>application/xml</td>
<td>Plain old xml</td>
<td>org.apache.axis2.transport.http.ApplicationXMLFormatter</td>
</tr>
<tr>
<td>text/xml</td>
<td>Soap envelope</td>
<td>org.apache.axis2.transport.http.SOAPMessageFormatter</td>
</tr>
<tr>
<td>application/soap+xml</td>
<td>Soap envelope</td>
<td>org.apache.axis2.transport.http.SOAPMessageFormatter</td>
</tr>
<tr>
<td>application/json</td>
<td>JSON (mapped notation)</td>
<td>org.apache.axis2.json.gson.JsonFormatter</td>
</tr>
</tbody>
</table>
Working with RDBMS

Given below are some advanced configurations that you can use when you define a data service using an RDBMS as the datasource.

- **Datasource Configuration Parameters**
- **Dynamic User Authentication**
- **Securing Datasource Password**
- **Tuning Datasource Configuration Parameters**

**Datasource Configuration Parameters**

When DSS processes a database operation, it spawns a database connection from an associated datasource. After using this connection, the DSS returns it to the pool of connections. The physical connection is not dropped with the database server unless it becomes stale or the datasource connection is closed. This is called **datasource connection pooling** and is a recommended way to gain more performance/throughput in the system. RDBMS datasources in DSS 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 management console when creating a datasource. Click and expand the section shown below:

Each parameter is described below:

The default values of the following parameters might not be optimal for the specific hardware/server configurations in your environment. We recommend you to carry out load tests in your environment to tune them accordingly. See **Tuning Datasource Configuration Parameters** for information on how you can select...
the best values for these parameters.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Isolation</td>
<td>The default TransactionIsolation state of connections created by this pool.</td>
</tr>
</tbody>
</table>
|                     | • TRANSACTION_UNKNOWN  
|                     | • TRANSACTION_NONE  
|                     | • TRANSACTION_READ_COMMITTED  
|                     | • TRANSACTION_READ_UNCOMMITTED  
|                     | • TRANSACTION_REPEATABLE_READ  
<p>|                     | • TRANSACTION_SERIALIZABLE  |
| Initial Size        | (int)                                                                                                                                 |
|                     | The initial number of connections created when the pool is started. Default value is 0.                                                        |
| Max. Active         | (int)                                                                                                                                 |
|                     | The maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.                     |
|                     | If you set this value too low, the response times for some requests might slow down as they have to wait for connections to get free. A value too high might cause too much memory and resource utilization and the system to slow down or be unresponsive. |
| Max. Idle           | (int)                                                                                                                                 |
|                     | The maximum number of connections that can remain idle in the pool, without extra ones being released. Default value is 8. Put a negative value for unlimited. Idle connections are checked periodically (if enabled) and connections that have been idle for longer than minEvictableIdleTimeMillis will be released. (also see testWhileIdle) |
| Min. Idle           | (int)                                                                                                                                 |
|                     | 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. (also see testWhileIdle) |
| 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). |</p>
<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>validationQueryTimeout</td>
<td>(int)</td>
<td>The timeout in seconds before a connection validation queries fail. This works by calling <code>java.sql.Statement.setQueryTimeout(seconds)</code> on the statement that executes the <code>validationQuery</code>. The pool itself doesn't timeout the query. It is still up to the JDBC driver to enforce query timeouts. A value less than or equal to zero will disable this feature. The default value is −1.</td>
</tr>
<tr>
<td>Test On Return</td>
<td>(boolean)</td>
<td>Used to indicate if objects will be validated before returned to the pool. NOTE - for a true value to have any effect, the validationQuery parameter must be set to a non-null string. The default value is false.</td>
</tr>
<tr>
<td>Test On Borrow</td>
<td>(boolean)</td>
<td>Used to indicate if objects will be validated before borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. NOTE - for a true value to have any effect, the validationQuery parameter must be set to a non-null string. In order to have a more efficient validation, see <code>validationInterval</code>. Default value is false.</td>
</tr>
<tr>
<td>Test While Idle</td>
<td>(boolean)</td>
<td>The indication of whether objects will be validated by the idle object evictor (if any). If an object fails to validate, it will be dropped from the pool. NOTE - for a true value to have any effect, the validationQuery parameter must be set to a non-null string. The default value is false and this property has to be set in order for the pool cleaner/test thread to run (also see <code>timeBetweenEvictionRunsMillis</code>).</td>
</tr>
<tr>
<td>Time Between Eviction Runs Mills</td>
<td>(int)</td>
<td>The number of milliseconds to sleep between runs of the idle connection validation/cleaner thread. This value should not be set under 1 second. It dictates how often we check for idle, abandoned connections, and how often we validate idle connections. The default value is 5000 (5 seconds).</td>
</tr>
<tr>
<td>Number of Tests Per Eviction Run</td>
<td>(int)</td>
<td>The number of objects to examine during each run of the idle object evictor thread.</td>
</tr>
<tr>
<td>Minimum Evictable Idle Time</td>
<td>(int)</td>
<td>The minimum amount of time an object may sit idle in the pool before it is eligible for eviction. The default value is 60000 (60 seconds).</td>
</tr>
</tbody>
</table>
Remove Abandoned (boolean)
Flag to remove abandoned connections if they exceed the removeAbandonedTimeout. If set to true a connection is considered abandoned and eligible for removal if it has been in use longer than the removeAbandonedTimeout. Setting this to true can recover db connections from applications that fail to close a connection. See also 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 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 borrow because a stack trace has to be generated. The default value is false.

Auto Commit (boolean) The default auto-commit state of connections created by this pool. If not set, default is JDBC driver default (If not set then the setAutoCommit method will not be called.)

Default Read Only (boolean) The default read-only state of connections created by this pool. If not set then the setReadOnly method will not be called. (Some drivers don't support read only mode, ex: 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 org.apache.tomcat.jdbc.pool.Validator interface and provides a no-arg constructor (may be implicit). If specified, the class will be used to create a Validator instance which is then used instead of any validation query to validate connections. The default value is null. An example value is com.mycompany.project.SimpleValidator.

Connection Properties (String) The connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be [propertyName=property;]*

Init SQL The 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) 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.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fair Queue</strong></td>
<td>(boolean) Set to true if you wish that calls to getConnection should be treated fairly in a true FIFO fashion. This uses the org.apache.tomcat.jdbc.pool.FairBlockingQueue 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 fairQueue=true there is a decision making process based on what operating system the system is running. If the system is running on Linux (property os.name=Linux. To disable this Linux specific behavior and still use the fair queue, simply add the property org.apache.tomcat.jdbc.pool.FairBlockingQueue.ignoreOS=true to your system properties before the connection pool classes are loaded.</td>
</tr>
<tr>
<td><strong>Abandon When Percentage Full</strong></td>
<td>(int) Connections that have been abandoned (timed out) won't get closed and reported up unless the number of connections in use are above the percentage defined by abandonWhenPercentageFull. The value should be between 0-100. The default value is 0, which implies that connections are eligible for closure as soon as removeAbandonedTimeout has been reached.</td>
</tr>
<tr>
<td><strong>Max Age</strong></td>
<td>(long) Time in milliseconds to keep this connection. When a connection is returned to the pool, the pool will check to see if the now - time-when-connected &gt; maxAge has been reached, and if so, it closes the connection rather than returning it to the pool. The default value is 0, which implies that connections will be left open and no age check will be done upon returning the connection to the pool.</td>
</tr>
<tr>
<td><strong>Use Equals</strong></td>
<td>(boolean) Set to true if you wish the ProxyConnection class to use String.equals and set to false when you wish to use == when comparing method names. This property does not apply to added interceptors as those are configured individually. The default value is true.</td>
</tr>
<tr>
<td><strong>Suspect Timeout</strong></td>
<td>(int) Timeout value in seconds. Default value is 0. Similar to to the removeAbandonedTimeout value but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if logAbandoned is set to true. If this value is equal or less than 0, no suspect checking will be performed. Suspect checking only takes place if the timeout value is larger than 0 and the connection was not abandoned or if abandon check is disabled. If a connection is suspect a WARN message gets logged and a JMX notification gets sent once.</td>
</tr>
<tr>
<td><strong>Alternate User Name Allowed</strong></td>
<td>(boolean) By default, the jdbc-pool will ignore the DataSource.getConnection(username,password) call, and simply return a previously pooled connection under the globally configured properties username and password, for performance reasons. The pool can however be configured to allow use of different credentials each time a connection is requested. To enable the functionality described in the DataSource.getConnection(username,password) call, simply set the property alternateUsernameAllowed to true. Should you request a connection with the credentials user1/password1 and the connection was previously connected using different user2/password2, 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.</td>
</tr>
</tbody>
</table>


**Dynamic User Authentication**

Dynamic user authentication allows you to authenticate database users dynamically for each data service call. The Data Services Server implements it using a mapping between the server users and the database users. This
mapping can be either,

- Static inside the data service configuration.
- Provided at runtime through a Java class that implements the interface `org.wso2.carbon.dataservices.core.auth.DynamicUserAuthenticator`.

The following topics explain both:

- Static configuration
- Runtime configuration
- Dynamic user lookup order of precedence
- Use of external datasources

Static configuration

You can specify a code as shown in the following example in the datasource configuration section of the data service.

```xml
<data name="RDBMSSample" serviceGroup="RDBMS">
  <config id="default">
    <property name="driverClassName">org.h2.Driver</property>
    <property name="url">jdbc:h2:file:./samples/database/DATA_SERV_SAMP</property>
    <property name="username">wso2ds</property>
    <property name="password">wso2ds</property>
    <property name="dynamicUserAuthMapping">
      <configuration>
        <entry request="admin">
          <username>wso2ds</username>
          <password>wso2ds</password>
        </entry>
        <entry request="user1">
          <username>dbuser1</username>
          <password>dbpass1</password>
        </entry>
        <entry request="*">
          <username>guest</username>
          <password>guest</password>
        </entry>
      </configuration>
    </property>
  </config>
  ....
</data>
```

The configuration above maps the two Carbon users to specific database credentials and the rest of the users to a different username/password pair. The `dynamicUserAuthMapping` property in `configuration/entry/@req`
User represents the incoming Carbon user, and the username and password elements that follow represent the mapped database credentials.

For dynamic user authentication to work, security must be enabled in the data service through UsernameToken for user authentication. If user authentication is not available when a dynamicUserAuthMapping section is specified, it maps to the request="*" scenario by default.

The following screenshot shows a sample configuration of dynamic user mappings. For each entry, the Carbon user and the target database user/password can be mapped.

**Runtime configuration**

In the runtime mode, the property `dynamicUserAuthClass` must be specified instead of the datasource configuration property `dynamicUserAuthMapping`. The `dynamicUserAuthClass` property's value must have the fully-qualified class name of a Java class that implements the interface `org.wso2.carbon.dataservices.core.auth.DynamicUserAuthenticator`. The interface is as follows:
public interface DynamicUserAuthenticator {
    /**
     * This method is used to lookup a username/password pair given a
     * source username.
     * @param user The source username
     * @return A two element String array containing the username and
     * password respectively
     * @throws DataServiceFault
     */
    String[] lookupCredentials(String user) throws DataServiceFault;
}

The following example code snippet shows an implementation of a dynamic user authenticator class.

```java
package samples;
import org.wso2.carbon.dataservices.core.DataServiceFault;
import org.wso2.carbon.dataservices.core.auth.DynamicUserAuthenticator;

public class MyDynAuthClass implements DynamicUserAuthenticator {
    @Override
    public String[] lookupCredentials(String user) throws DataServiceFault {
        if ("admin".equals(user)) {
            return new String[] {"wso2ds", "wso2ds"};
        } else if ("user1".equals(user)) {
            return new String[] {"dbuser1", "dbpass1"};
        } else if ("user2".equals(user)) {
            return new String[] {"dbuser2", "dbpass2"};
        } else {
            throw new DataServiceFault("The user '" + user + '" not supported in invoking the target data service");
        }
    }
}
```

The `lookupCredentials` method takes in the request user and returns the database username/password in a String array. The `dbs` file configuration format is as follows:
The dynamic user authentication class can be specified in the field shown in the screenshot below.

Dynamic user lookup order of precedence

In a single datasource configuration, both the static and the runtime configurations can be available at once. The server processes them as follows:

- Higher precedence goes to the static mapping in initially looking up the credentials. The "***" request setting is ignored in the first pass
- If a request user/database credentials mapping cannot be found, the secondary runtime Java class implementation is used to look up the user
- If the previous option also fails, the program returns for the primary static mapping and processes the "***" request mapping
- The data service request returns an error only if all of the above options fail
Use of external datasources

When using datasources that are not inline like Carbon, JNDI etc. the datasources must be specified in a way that its connections can be created for selected users. Specifically in Carbon datasources, enable the setting alternateUsernameAllowed for dynamic user authentication to function.

Securing Datasource Password

All WSO2 products, by default, come with a secure vault implementation, which is a modified version of synapse secure vault. It provides capability to securely store-sensitive data such as plain-text passwords in configuration files of the WSO2 Carbon platform, such as user-mgt.xml, Carbon.xml, Axis2.xml, registry.xml etc. All WSO2 Carbon-based products inherit the secure vault implementation from the core Carbon platform. For more information, see the section on working with security.

However, when securing passwords of more product-specific configuration files such as data service configurations, the steps may vary. Therefore, WSO2 DSS provides the feature to securely store sensitive data in data service configuration files, using the secure vault functionality. Users can encrypt their passwords using tokens instead of the actual password inside the data service configuration file. See the instructions given below.

1. Run ciphertool script from `<PRODUCT_HOME>/bin` directory.
   - Linux: sh ciphertool.sh -Dconfigure
   - Windows: ciphertool.bat -Dconfigure
2. To encrypt the plain text using ciphertool, run the ciphertool script again without -Dconfigure option. It asks for the KeyStore Password of the running Carbon instance. The default is wso2carbon.
3. Then provide the plain text value that needs to be encrypted and the tool returns the encrypted text value.
4. Update the `<PRODUCT_HOME>/repository/conf/security/cipher-text.properties` file by adding a new alias (any name of your preference) and the encrypted value.
5. In the Edit datasource window of the management console select Use as Secret Alias option. In the Password field, provide the alias name instead of the actual password.

   ![Add New Data Source](image)

6. The namespace and alias will be added to the .dbs file as follows:

   ```xml
   <data name="RDBMS Sample" enableBcaring="true" serviceGroups="RDBMS" xmlns:svns="http://org.wso2.securevault/configuration">
     <config id="default">
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.protocol" value="http"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/>
       <property name="org.wso2.ws.dataservice.driver" value="org.h2.Driver"/><property>
       </config>
   </data>
   ```
Tuning Datasource Configuration Parameters

When you configure datasource connections, ensure that the parameters relevant to connection pooling are tuned according to your production environment. Consider the following when tuning:

- The application's concurrency requirement.
- The average time taken to run a database query.
- The maximum number of connections the database server can support.

The goal of tuning the pool properties is to maintain a pool that is large enough to handle the peak load to the datasource without unnecessarily utilising resources. The following are the highly used, important properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>How to Tune</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. Idle</td>
<td>Configure this to match the expected minimum concurrency. The default value is 0.</td>
</tr>
<tr>
<td>Max. Idle</td>
<td>The value should be less than the Max. Active value. For high performance, tune Max. Idle to match the number of average, concurrent requests to the pool. If this value is set to a large value, the pool will contain unnecessary idle connections.</td>
</tr>
</tbody>
</table>
| Max. Active   | The maximum latency (approximately) = \((P / M) * T\),  

\[
\text{where,} \\
M = \text{Max. Active value} \\
P = \text{Peak concurrency value} \\
T = \text{Time (average) taken to process a query.}
\]

Therefore, by increasing the Max. Active value (up to the expected highest number of concurrency), the time that requests wait in the queue for a connection to be released will decrease. But before increasing the Max. Active value, consult the database administrator, as it will create up to Max.Active connections at burst times from a single node, and it may not be possible for the DBMS to handle the accumulated count of these active connections. |
| Max. Wait     | Adjust this to a value slightly higher than the maximum latency for a request. That is, Max. Wait = \((P / M) * T + \text{buffer time.}\) |
| Validation Query | Set "Validation Query" to a simple test query like SELECT 1. |
| Test On Borrow | When the connection to the database is broken, the connection pool does not know that the connection has been lost. As a result, the connection pool will continue to distribute connections to the application until the application actually tries to use the connection. To resolve this problem, set "Test On Borrow" to "true" and make sure that the "Validation Query" property is set. |
| validationInterval | This parameter allows to control how frequently a given validation query is executed. By default it is set to 30 seconds. Deciding the value for the "validationInterval" depends on the target application behavior. If a larger value is set, the frequency of executing the Validation Query is low, which results in better performance (but for values larger than several seconds, this becomes negligible, as the validation query usually executes very fast). With a smaller value, a stale connection will be identified quickly when it is presented. This maybe required for systems where the connections should be instantly repaired in a case like a database server restart. Therefore, selecting a value for the this property is a trade-off and ultimately depends on what is acceptable for the application. |
The following properties are used for detecting and removing the connections that get leaked from the connection pool:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>How to configure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove Abandoned</td>
<td>If this property is set to 'true', a connection is considered abandoned and eligible for removal if it has been in use for longer than the removeAbandonedTimeout value explained below.</td>
</tr>
<tr>
<td>Remove Abandoned</td>
<td>The time in seconds that should pass before a connection that is in use can be removed. This is the time period after which the connection will be declared abandoned. This value should be set to the longest running query that the applications might have.</td>
</tr>
<tr>
<td>Log Abandoned</td>
<td>Set this property to &quot;true&quot; if you wish to log when the connection was abandoned. If this option is set to &quot;true&quot;, a stack trace is recorded during the dataSource.getConnection call and is printed when a connection is not returned.</td>
</tr>
</tbody>
</table>

**Writing Advanced Data Service Queries**

When you define a data service, you will be defining a query specifying how to interact with a particular datasource. Follow the tutorials for instructions on how to write a data service query. The following topics will guide you on how to write advanced queries for data services.

- Defining Input and Output Mappings
- Adding an SQL Dialect
- Advanced Query Properties
- Calling MySQL or Oracle Functions in a Query
- Data Types of Mappings
- Defining a Dynamic SQL Query
- Defining Named Parameters
- Generating the Output of a Query
- Grouping Data into Complex Elements
- Handling Events
- Exposing Data in JSON Format
- Exposing Relational Data in RDF Format
- Sample Queries

**Defining Input and Output Mappings**

This section describes few key features useful when defining Input and Output Mappings in queries.

- Adding Input Mappings
- Adding Output Mappings

**Adding Input Mappings**

Input mappings allow you to add parameters to a query so that you can set the parameter value when executing the query. For example, when you define a query as `SELECT *, FROM TEST_TABLE, WHERE ID=1`, an input mapping is a parameter that sets the value of ID.

- Generating input mappings
- Adding input mappings manually
- Returning generated keys
- Returning updated row count
Generating input mappings

If you have defined an SQL query, you can generate input mappings corresponding to the input fields specified in the query by clicking Generate Input Mappings. As shown in the example below, an input mapping is created for the `emp_no` field, which will allow you to invoke this query by specifying a value for this field as an input.

Adding input mappings manually

If you are manually adding input mappings, click Add New Input Mappings to open the following page:

The following options are available when you create an input mapping:

- Parameter type
Parameter type
This is the data type of the input mapping, which determines how the input mapping parameter will be given in the target query.

- **SCALAR**: In the target query, the parameter will be used as one value.
- **ARRAY**: In the target query, the parameter will contain one or many values for a mapped parameter.

Note that ARRAY parameter type cannot be used with the QUERY_STRING data type (SQL type).

In the context of RDBMS and SQL datasources, an ARRAY parameter mapped to an SQL query will be expanded to multiple comma separated parameters at runtime. For example, this can be used in SQL statement conditions such as SELECT ... WHERE ... IN(?).

**SQL type**
The data type of the corresponding SQL parameter can be selected from this menu. Note that the QUERY_STRING data type cannot be used if the **parameter type** is set to ARRAY. Find more from here about data types.

**Default value**
Default values help you automatically assign a value to a parameter when a user has not entered a specific parameter value in a request. Default values are added when defining queries. Therefore, this value gets automatically added to the query if it is ignored by the user.

You can refer to Internal Property Values using Default Values. You can use special system variables that are defined as default values. At the moment, it only provides a variable for retrieving the username of the current user authenticated in a secured data service. You can access this variable as follows:

\[\text{#{USERNAME}}\]: Dynamically replaces the input mapping with the current user's username when a data service request is processed.

\[\text{#{NULL}}\]: Sets the current input mapping value to null. It's the same as providing "xsi:nil" in the incoming message's input parameter element.

\[\text{#{TENANT_ID}}\]: Represents the current tenant ID. This is useful in a Stratos deployment where multiple tenants live in the same server.

\[\text{#{USER_ROLES}}\]: This value contains the list of user roles that the current calling user has. If the parameter mapped is of type ARRAY, it will have the full list of user roles. If it's a SCALAR, it will only contain the first user role of the user.

For a demonstration of the usage of default values, see Default Values Sample.

**IN/OUT Type**
These are used in stored procedures which takes out parameters and in/out parameters. IN is the usual parameter
we give to provide some value inside. OUT only returns a value from a stored procedure. INOUT does both.

Validators

Validators are added to individual input mappings in a query. Input validation allows data services to validate the input parameters in a request and stop the execution of the request if the input doesn’t meet required criteria. WSO2 Data Service Server provides a set of built-in validators for some of the most common use cases. It also provides an extension mechanism to write custom validators.

- Long Range validator
- Double Range validator
- Length validator
- Pattern validator
- Custom validators

Long Range validator

Validates if an integer value is in the specified range. The validator requires a minimum and a maximum value to set the range. For example,

Double Range validator

Validates if a floating point is in the specified range. The validator requires a minimum and a maximum value to set the range. For example,
Length validator

Validates the string length of a given parameter against a specified length. For example,
Pattern validator

Validates the string value of the parameter against a given regular expression. For example,

![Pattern validator](image)

Custom validators

Used to add your own validation logic by implementing the interface `org.wso2.carbon.dataservices.core.validation.Validator`. The definition of the interface is as follows:

```java
public interface Validator {
    public void validate(ValidationContext context, String name, ParamValue value) throws ValidationException;
}
```

If the validation fails, the validate method in the interface by default throws an exception of type `ValidationException`. The parameters of the method are as follows:

- `context`: Is of type `ValidationContext`, which contains information about the full set of parameters passed into the request. When the validation logic depends on other parameters, the validation context can be used to check the names/values of the rest of the parameters.
- `name`: A string value that represents the name of the parameter to be validated.
- `value`: Is of type `ParamType`, which represents the value of the parameter to be validated. It is either `SCALAR` or `ARRAY`.

If you need to provide properties when initializing the custom validator, it is necessary to implement the `org.wso2.carbon.dataservices.core.validation.ValidatorExt` interface. This extends the `Validator` interface as shown below.
The init method initializes the set of properties provided for the custom validator via the management console or the configuration file of the data service. See the examples given below.

- Properties provided via the management console:

```
public interface ValidatorExt extends Validator {
    public void init(Map<String, String> props);
}
```

- Properties provided via the data service configuration file:

```
<param name="email" sqlType="STRING">
  <validateCustom class="org.acme.EmployeeEmailValidator">
    <properties>
      <property name="company">acme</property>
      <property name="domain">org</property>
    </properties>
  </validateCustom>
</param>
```

After creating a custom validator class, package it in a JAR file and store it in the server's classpath location for external libraries (which is the `<DSS_HOME>/repository/component/lib` directory)

**Returning generated keys**

The **Return Generated Keys** option appears under **Input Mappings** on the **Queries** page.

It inserts data to a table that has auto increment key columns. The auto incremented key value of the record is mapped to the result output mappings of the data service. For example, the sample query below is used to insert values to a table by the name `wes_teams`, which has an auto increment column:

```
INSERT INTO wes_teams(TEAM) VALUES(?)
```
Once the user selects Return Generated Keys option, an auto increment key is added as an output mapping as follows:
Returning updated row count

With the current functionalities of DSS, we don't have a way to indicate that the update operation did not affect any rows. But, we can return the updated row count as a response to the client in queries like update/insert to indicate how many rows are affected by the query execution.

- Return Generated Keys
- Return Updated Row Count

Adding Output Mappings

Just as Input mapping allows you to add parameters to a query, output mapping determines how the output of a query should be presented.

- Generating output mappings
- Adding output mappings manually

Generating output mappings

Note that this option is only available for SELECT statements excluding SELECT *, and for datasources such as RDBMS.
If you have defined an SQL query, you can generate output mappings corresponding to the fields specified in the query by clicking Generate Response. In the example shown below, there is an SQL query that needs to output values for the customernumber and customername fields in the customers table.

When you click Generate Response, output mappings will be created automatically for the specified fields as shown below.

Adding output mappings manually

If you are manually adding output mappings, you need to update the Result (Output Mapping) section in the Queries page shown below. Once you have entered the main output mapping options, click Add New Output Mapping to open the Edit Output Mapping page.
The following options are available for creating an output mapping:

- **Set Output Type**
- **Use column numbers**
- **Escape non printable characters**
- **Row namespace**
- **Query result export**
- **Access control by user roles**

### Set Output Type

The output type determines the format in which the query output will be presented. You can select either XML, RDF or JSON. For details of how you can configure JSON as the output type go to Exposing Data in JSON Format.

### Use column numbers

If this option is selected the mapping will be done by the column number basis instead of the column name. The following screen shot provides an example for using column numbers.

### Escape non printable characters

Tick this option if the data in your database consists of characters that are not serializable to XML. Few examples are & < > " ". When you invoke services that access such data and produce responses, the server throws errors. Ticking this option ensures that non-printable characters will be ignored when producing the responses.

### Row namespace

See Defining Namespaces.

### Query result export
When you click **Add New Output Mapping** in the **Result(Output Mapping)** section of the **Queries** page as explained above, the **Edit Output Mapping** page will open. You can specify the type of fields that will present the output of your query by giving the data source type, output field name, data source column name etc.

In the **Edit Output Mapping** page, you can define query result export options. Query Request Export feature must be used in conjunction with boxcarring. It allows individual queries executed in a boxcarring session to communicate with each other. The concept is 'exporting' a specific result element so that the next calling query will get that result element as a query parameter. So, if you've two queries, namely, 'query1' and 'query2' that's executed sequentially in a boxcarring session, and if 'query1' has a specific result element and that element is exported with the name 'foo', then 'query2' also gets a query param named 'foo'. So when this boxcarring session is executed, the query1’s exported value will be passed into query2 as an input parameter.

This feature is very useful in situations where the result of an earlier-executed query is required for the execution of a subsequent query (e.g. a newly created primary key).

The following figure shows how a result element can be declared to be exported with a given name when defining a query in a data service.

There are two export types that can be used.

- **SCALAR**: The single element value is exported. If there are multiple instances of this value, the last one will be exported.
- **ARRAY**: An array of values will be exported. Each occurrence of the value is added to an array and exported.

For a demonstration on the usage of export options, refer to **Boxcarring Sample**.

**Access control by user roles**

In the **Edit Output Mapping** page, you can also define the access controls for the output mapping by assigning user roles. This ensures that only the users assigned to that specific user role are allowed to view the particular information in the query output. Go to **WSO2 administration guide** to see how user roles and their permissions are defined in your system.
As shown below, the user roles available in your system will be listed. You can select the required user roles for the output mapping.

User roles you selected will be assigned to the output mapping as shown below.

Adding an SQL Dialect

SQL Dialect support allows users to create multiple SQL queries for different SQL dialects by checking the driver. This helps maintain driver-specific functionality/key words in SQL statements of a single data service query.
For example, to determine the data length retrieved from a column, there can be different ways to write the SQL query depending on the SQL driver.

**On MySQL and PostgreSQL:**

```sql
SELECT OCTET_LENGTH(employeeNumber)
FROM Employees;
```

**On Microsoft SQL Server:**

```sql
SELECT DATALENGTH(employeeNumber)
FROM Employees
```

**On Oracle:**

```sql
SELECT LENGTH(employeeNumber)
FROM Employees
```

To avoid writing different data service queries for the same operation depending on the configuration, write all three SQL queries in the same data service query as shown below.

```xml
<config id="default">
  <property name="org.wso2.ws.dataservice.driver">com.mysql.jdbc.Driver</property>
  <property name="org.wso2.ws.dataservice.protocol">jdbc:mysql://localhost:3306/MyDB</property>
  <property name="org.wso2.ws.dataservice.user">root</property>
  <property name="org.wso2.ws.dataservice.password">password</property>
</config>

<query id="ql" useConfig="default">
  <sql>SELECT DATALENGTH(employeeNumber) FROM Employees</sql>
  <sql dialect="oracle">SELECT LENGTH(employeeNumber) FROM Employees</sql>
  <sql dialect="h2,mysql,postgresql">SELECT OCTET_LENGTH(employeeNumber) FROM Employees</sql>
  <sql dialect="mysql">SELECT DATALENGTH(employeeNumber) FROM Employees</sql>
</query>
```

Follow the steps below to add SQL dialects to a query.

1. The SQL dialect option appears when adding a query to datasources such as RDBMS. For example,
2. Click **Add New SQL Dialect** to open the **SQL Dialect** window. Select the required SQL driver and define the SQL statement as shown below. If the SQL statement should be the same for multiple drivers, select all drivers (e.g., MySQL, PostgreSQL and H2) at once and define the statement as shown below.
To define an SQL dialect for a driver other than the ones listed in the supported drivers list, provide the driver prefix in the text field and define the SQL query as shown above.

**Advanced Query Properties**

Advanced query properties help define additional features when querying the database. This option is available when adding queries to data sources such as RDBMS.

Query property details are described below.
### Property Name | Description
--- | ---
Timeout | Sets a timeout for the underlying JDBC query.
Fetch Direction | Forward - rows in a result set will be processed in a forward direction; first-to-last. Backward - the rows in a result set will be processed in reverse direction; last-to-first.
Fetch Size | The number of rows that should be fetched from the database when more rows are needed. If the fetch size is zero, the JDBC driver ignores the value and is free to make its own best guess as to what the fetch size should be. Note that the fetch size is set to a lower value in the DSS by default. However, if you expect a very large number of rows to be fetched, you should increase the fetch size accordingly (e.g. 1000) to improve performance.
Max Field Size | Maximum data size for the field. Used to reduce the size each field takes in order to eliminate the possibility of hitting a db limit.
Max Rows | Maximum number of rows to be returned. Zero means all rows.
Force Stored Procedure | Forces the current SQL statement as a stored procedure.
Force JDBC Batch Requests | Forces to use native JDBC batch request.

### Calling MySQL or Oracle Functions in a Query

Given below is how to add a MySQL/Oracle function call to a data service.

- Calling a MySQL function
- Calling an Oracle function

### Calling a MySQL function

Assume you have the following MySQL function, which takes a string parameter and returns the same as output. Create a database before executing the query.

```sql
create function myFunction(p_inparam varchar(20))
returns varchar(20)
begin
    declare output_text varchar(20);
    set output_text = p_inparam;
    return output_text;
end
```
To call this function from the data service, add a query to the data service definition file (.dbs) pointing to an RDBMS datasource that connects to the MySQL database. For example,

<sql>select myFunction('WSAS') as ABC</sql>

For example, see the following data service configuration:

```xml
<data name="sqlfunctionService">
  <config id="mynew">
    <property name="driverClassName">com.mysql.jdbc.Driver</property>
    <property name="url">jdbc:mysql://localhost:3306/sample</property>
    <property name="username">root</property>
    <property name="password">root</property>
  </config>
  <query id="NewfunctionQuery" useConfig="mynew">
    <sql>select myFunction('WSAS') as ABC</sql>
    <result element="wsas" rowName="wsas">
      <element column="output_text" name="n_param" xsdType="string"/>
    </result>
    <param name="imparam" sqlType="STRING"/>
  </query>
  <operation name="functionop">
    <call-query href="NewfunctionQuery">
      <with-param name="imparam" query-param="imparam"/>
    </call-query>
  </operation>
</data>
```

You can also call this function in the Query Details page of the management console as follows:

![Query Details](image)

**Calling an Oracle function**

Assume you have the following Oracle stored function, which returns the total number of entries in a table:
CREATE OR REPLACE FUNCTION myfunction(ename IN VARCHAR, eid IN NUMBER) 
RETURN INTEGER 
AS myCount INTEGER;
BEGIN 
    INSERT INTO TEAMS values(eid, ename); 
    SELECT COUNT(*) into myCount from TEAMS; 
    RETURN myCount; 
END;
/

Create a table before executing the query as follows:

CREATE TABLE TEAMS(id INTEGER, team VARCHAR(30));

To call this function from the data service, add a query to the data service definition file (.dbs). For example,

"{call ?:myfunction(?,?)}"

First input parameter carries the return value of the function. Other two parameters are inputs to the function. You must define an Input parameter with OUT type to get the result of function (i.e., the first parameter in the query above). Then, define a Output parameter to get this value as a result set from the data service. The following code segment does this:

    <result element="TotalTeams" rowName="">
        <element name="totalTeams" column="totalTeams" xsdType="xs:integer" />
    </result>
    <param name="totalTeams" sqlType="INTEGER" type="OUT" ordinal="1" /> 

For example, see the following data service configuration:
You can call this function in the **Query Details** page of the management console as before.

### Data Types of Mappings

This section describes few important points on the following data types used when defining Input and Output Mappings:

- **Array data type**
- **Binary data type**
- **User-defined data types**

#### Array data type

Array parameter type enables data services to dynamically provide a set of values without knowing the size of the data set beforehand. Generally, more than single data types enable building queries for more complex inputs. Array data type is available as a parameter type when creating input mappings queries that are defined while creating data services.

Let's see how array types are handled internally in the SOAP message-level. When the parameters are serialized (in scalar types), there is only one element per parameter in the SOAP message. In arrays, the element representing the parameter is repeated. For example, refer to the sample SOAP message below, which contains the array type `employeeNumbers`.

```xml
<data name="testOracleFunction">
  <config id="or">
    <property name="org.wso2.ws.dataservice.driver">oracle.jdbc.driver.OracleDriver</property>
    <property name="org.wso2.ws.dataservice.protocol">jdbc:oracle:thin:user/pwd@localhost:1521/XE</property>
    <property name="org.wso2.ws.dataservice.user">user</property>
    <property name="org.wso2.ws.dataservice.password">pwd</property>
  </config>
  <query id="q1" useConfig="or">
    <sql>{call ?:=myfunction(?,?)}</sql>
    <result element="TotalTeams" rowName="">
      <element name="totalTeams" column="totalTeams" xsdType="xs:integer" />
    </result>
    <param name="totalTeams" sqlType="INTEGER" type="OUT" ordinal="1" />
    <param name="ename" sqlType="STRING" ordinal="2" />
    <param name="eid" sqlType="INTEGER" ordinal="3" />
  </query>
  <operation name="op1">
    <call-query href="q1">
      <with-param name="ename" query-param="ename" />
      <with-param name="eid" query-param="eid" />
    </call-query>
  </operation>
</data>
```
Array types are properly mentioned in the WSDL generation. As a result, it is suitably presented in a service that is code generated.

Note that ARRAY parameter type cannot be used with the QUERY_STRING data type (SQL type).

Binary data type

When using data services, you might across the need to transfer binary data from/to the server. This is handled using Base64 encoding. When sending, binary data must be encoded in Base64 format. Similarly, when receiving, the receiving character data must be Base64-decoded in order to retrieve the original binary data back.

Binary data type is available as an SQL Type when creating input mappings for your queries. For example,

![Add Input Mapping (dataservice1/query1)](image)
When defining the result of a data service, in order to declare that a binary value is expected, add a suitable entry in Output Mappings. The **Schema Type** of the Output Mapping should be selected as "xs:base64Binary." For example,

![Add Output Mapping](image)

When using code-generated clients, the encoding/decoding of Base64 data need not be explicitly done by the user since the existence of binary data is mentioned in the WSDL. For example, in Axis2 code-generated stubs, the binary types are handled using the "DataHandler" class.

**User-defined data types**

Apart from the standard data types (such as Varchar, Integer etc.) the 'Data Service Hosting' feature also supports querying of custom objects, which are usually called User Defined Types (UDT). Users can query UDTs with ordinary SQL queries as well as OUT parameters of stored procedures.

**Query UDTs with Ordinary SQL Queries**

In this option, you are only required to define an output mapping corresponding to the UDT to be queried. An important point is that the attributes of a UDT is queried depending on the order they are specified in the UDT. For example, if the UDT carries the structure below, the attribute index "0" maps to the attribute "Id".

```
"SampleUDT{Id Integer, Name Varchar(100)}"
```

Similarly, attribute "Name" can be retrieved via the index "1". The following image depicts how it is done in the Data Services Wizard:
It is possible to retrieve the values of UDTs via the OUT parameters defined in stored procedures. The user should define an input mapping as an OUT parameter as well as an output mappings, in order to actually retrieve the values of the UDT attributes.

Shown below is an example of how we can define the Input mapping in order to register an OUT parameter to be able to retrieve the value of a sample UDT called "PERSON_T".
Defining a Dynamic SQL Query

Dynamic SQL query support allows you to change SQL queries (e.g., defining additional conditions in the SQL) in the runtime without changing the data service configuration. For this to work, you must specify required SQL query statements (e.g., with WHERE clause) as a QUERY_STRING data type. These statements will be directed to the final SQL query in the runtime.

Dynamic query support can lead to SQL injection attacks. Therefore, we recommend that the clients validate the values set to QUERY_STRING at runtime.

The QUERY_STRING data type is available as an SQL type when creating Input Mappings for queries. For example,
You can add the SQL query using the mapping name:

```xml
<query id="employeesSQL" useConfig="default">
  <property name="driverClassName">org.h2.Driver</property>
  <property name="url">jdbc:h2:file:./samples/database/DATA_SERV_SAMP</property>
  <property name="username">wso2ds</property>
  <property name="password">wso2ds</property>
  <property name="minIdle">1</property>
  <property name="maxActive">10</property>
  <property name="autoCommit">false</property>
</query>
```

A sample configuration for the data service is shown below:
<sql>select * from Employees :filterQuery</sql>

<result element="employees" rowName="employee">
    <element column="lastName" name="last-name" xsdType="string"/>
    <element column="firstName" name="first-name" xsdType="string"/>
    <element column="email" name="email" xsdType="string"/>
    <element column="salary" name="salary" xsdType="double"/>
</result>

<param name="filterQuery" sqlType="QUERY_STRING"/>

</query>

<query id="customerInCountrySQL" useConfig="default">
    <sql>select * from Customers where country = :country :filter</sql>
    <result element="customer-addresses" rowName="customer-address">
        <element column="customerNumber" name="customer-number" xsdType="integer"/>
        <element column="contactLastName" name="contact-last-name" xsdType="string"/>
        <element column="contactFirstName" name="contact-first-name" xsdType="string"/>
        <element column="addressLine1" name="address-line1" xsdType="string"/>
        <element column="addressLine2" name="address-line2" xsdType="string"/>
        <element column="city" name="city" xsdType="string"/>
        <element column="state" name="state" xsdType="string"/>
        <element column="postalCode" name="postal-code" xsdType="string"/>
        <element column="country" name="country" xsdType="string"/>
    </result>

    <param name="country" sqlType="STRING"/>
    <param name="filter" sqlType="QUERY_STRING"/>
</query>

<query id="insertUpdateQuery" useConfig="default">
    <sql>:query</sql>
    <param name="query" sqlType="QUERY_STRING"/>
</query>

<operation name="getEmployees">
    <call-query href="employeesSQL">
        <with-param name="filterQuery" query-param="filterQuery"/>
    </call-query>
</operation>

<operation name="getCustomersInCountry">
    <call-query href="customerInCountrySQL">
        <with-param name="country" query-param="country"/>
        <with-param name="filter" query-param="filter"/>
    </call-query>
</operation>

<operation name="insertUpdateOp">
    <call-query href="insertUpdateQuery">
        <with-param name="query" query-param="query"/>
    </call-query>
</operation>
Defining Named Parameters

Named Parameters enable reusability of parameters and reduce the complexity of database configurations. Named parameters are specified in SQL queries of data services.

A named parameter must have the same name as the input parameter along with a colon ':' in front. For example,

```
</call-query>
</operation>
</data>
```

Generating the Output of a Query

The Generate Response link associated with queries generates the output of a query. See Adding Output Mappings for instructions.

Grouping Data into Complex Elements

Complex Elements help represent data in a structured manner. For example, let's take a table containing customer information. There can be several columns that keep data related to the address of employees such as number, street, city, postal code. You can group them into one element called address using complex elements and present them in a more structured manner.

Following example illustrates how to use complex elements:
After defining the query, proceed to adding an Output Mapping. In output mappings, select the Mapping Type as complex element. Specify an Element Name and click the Add Nested Element button. For example,

![Output Mapping Example](image)

The added element appears under Complex Elements in the Output Mapping.

According to the figure below, addressline1, addressline2 and city are nested elements that come under address element.
Handling Events

See the tutorial on Receiving Notifications from Data Services for instructions on how to define events for a data service. You can find more information on event handling from the following:

Executing an event trigger

You can set an event trigger as an input or an output event trigger in a data service query. Let's look at the usage of the two approaches.

Input event trigger

When an input event trigger is applied to a query, the event trigger is evaluated when the parameters are received by the query. An event-trigger executes an XPath expression against an XML element. This XML element is built from the input parameters that are represented by the element of the parameters wrapped by an element with the name of the query.

For example, take the following query:
<query id="incrementEmployeeSalaryQuery" useConfig="default">
   <sql>update Employees set salary=salary+? where employeeNumber=?</sql>
   <param name="increment" paramType="SCALAR" sqlType="DOUBLE" type="IN" ordinal="1" />
   <param name="employeeNumber" paramType="SCALAR" sqlType="INTEGER" type="IN" ordinal="2" />
</query>

Assuming that the values of increment and employeeNumber are value1 and value2 respectively, the XML element to be evaluated with the XPath expression is as follows. Also note that this XML element doesn't not have any namespaces associated with it.

```xml
<incrementEmployeeSalaryQuery>
   <increment>value1</increment>
   <employeeNumber>value2</employeeNumber>
</incrementEmployeeSalaryQuery>
```

**Output event trigger**

In this case, the event trigger is evaluated when a specific query is returning its result. There isn't a specific creation of an XML element that is used with the XPath expression like in the input event trigger. But the full result XML is used to evaluate it. The result will be namespace qualified. Therefore, you must write the XPath expressions accordingly.

**Contents of event notification**

In the event notification messages, additional information about the event is added to the SOAP Envelope/Body element. The following code demonstrates this.

```xml
<data-services-event>
   <service-name>$SERVICE_NAME</service-name>
   <query-id>$QUERY_ID</query-id>
   <time>$TIME</time>
   <content>$CONTENT</content>
</data-services-event>
```

- $SERVICE_NAME : Name of the service from which the event originated
- $QUERY_ID : The id of the query that triggered the event
- $TIME : The date/time when the event occurred
- $CONTENT : The XML element generated in case of an input event trigger. It is used when executing the XPath expression. It contains the input parameters wrapped with the query id value. In the case of an output event trigger, it contains the full result XML
For a sample demonstration on how to use eventing, see Eventing Sample.

Exposing Data in JSON Format

JSON mapping support in data services is available as another result type in the output mappings section of the data services query. That is, you can specify JSON as the output type for the query and then provide the JSON template that should be used for service responses.

From DSS version 3.2.0 onwards, the default JSON message formatter/builder is implemented based on org.apache.axis2.json.gson, which supports some new features as explained below.

- Basic Mapping
- Extended Properties for Mapping Fields
- Nested Queries

Basic Mapping

The following is a sample JSON mapping of the data services results. Here, we can see the JSON template, which is the format that is expected to be returned from our query.

```json
{
   "customers": {
      "customer": [
         {
            "phone": "$phone",
            "city": "$city",
            "contact": {
               "customer-name": "$customerName",
               "contact-last-name": "$contactLastName",
               "contact-first-name": "$contactFirstName"
            }
         }
      ]
   }
}
```

As shown in the sample given above, the column name values that are expected in the query result should be referred to by the column name with the "$" prefix. E.g. "$customerName".

Also, the structure of the JSON template should follow some guidelines in order to be compatible with the result. These guidelines are:

- The top most item should be a JSON object. It cannot be a JSON array.
- For handling multiple records from the result set, the immediate child of the top most object can be a JSON array, and the array should contain only a single object.
- If only a single result is returned, the immediate child of the top most object can be a single JSON object.
- After the immediate child of the top most object, there cannot be other JSON arrays in the mapping.

All JSON responses will be returned as an array.
Extended Properties for Mapping Fields

In a basic JSON output mapping, we specify the field values that we expect in the query result. You can give additional properties to this field mapping, such as data type of the field, the possible content filtering user roles etc. These extended properties for the fields are given in parentheses, with a list of string tokens providing the additional properties, separated by a semi colon (";").

Data Types

The example given below shows how the ‘data type’ property is added to the ‘age’ field using parentheses "()".

```json
{ "age" : "$age(type:integer)" }
```

- The extended property ‘type’ is given along with the value ‘integer’.
- The property and the value are separated by a colon.
- Note that the possible values for data type are “integer”, “long”, “double” and “boolean”.

Content Filtering (Required Roles)

The example given below shows how the ‘data type’ extended property as well as the ‘required roles’ extended property is added to the ‘age’ field using parentheses "()".

```json
{ "salary" : "$salary(type:double;requiredRoles:hr,admin)" }
```

- The ‘requiredRoles’ property is added along with the values ‘hr’ and ‘admin’.
- Note that the two values for required roles property are separated by a comma.
- Also, the two extended properties are separated by a semicolon.

Nested Queries

Nested queries can be mentioned in the JSON mapping by giving the query details as a JSON object attribute. That is, the name of the target query to be called and the property value (the fields in the result mapped with the target query parameters) are included in the JSON mapping as the object attribute name. Example:
**JSON Mapping with Nested Queries**

```json
{
    "Offices": {
        "Office": [
            {
                "officeCode": "@officeCode(type:integer)",
                "city": "$city",
                "country": "$country",
                "phone": "$phone",
                "@employeesInOfficeSQL": "$officeCode->officeCode"
            }
        ]
    }
}
```

In the nested query mapping,

- The target query name is mentioned by prefixing the query name with "@". Note "@employeesInOfficeSQL" in the example given above.
- The parameter mapping is added to the query by giving the following values: The field name in the result prefixed by "$", and the name of the target query parameter.
- These two values in the parameter mapping are separated by "->". See "$officeCode->officeCode" in the example given above.
- Note that the target query name and the parameter mapping are separated by a colon as follows: "@employeesInOfficeSQL": "$officeCode->officeCode"

A JSON Mapping Sample in WSO2 Data Services Server can be found here: [JSON Mapping Sample](#).

---

**JSON Mapping in tenant mode**

To run this feature in tenant mode, remove the following content in the `axis2.xml` and `tenant-axis2.xml` files.

```xml
<messageBuilder contentType="application/json"
class="org.apache.axis2.json.gson.JsonBuilder" />
<messageFormatter contentType="application/json"
class="org.apache.axis2.json.gson.JsonFormatter" />
```

Then, uncomment the following content in the same files.

```xml
<messageFormatter contentType="application/json"
class="org.apache.axis2.json.JSONMessageFormatter"/>
<messageBuilder contentType="application/json"
class="org.apache.axis2.json.JSONOMBuilder"/>
```
Exposing Relational Data in RDF Format

In this tutorial, we went through the process of exposing data stored in an RDF file using a data service. In the following instructions, we will look at how relational data can be exposed in RDF format by configuring the relevant data service query. For example, let's see how to expose a CSV datasource as an RDF resource.

Before you begin

Create and deploy a data service using a CSV datasource. See the tutorial on exposing CSV data as a data service for instructions.

Exposing relational data in RDF format

Follow the steps given below.

1. In the management console, click List under Services to open the Deployed Services page.
2. Select the service created earlier using a CSV datasource to go to its dashboard.
3. In the CSV data service’s dashboard, click Edit Data Service.

   Specific Configuration
   
   ![Edit Data Service (Wizard)]
   ![Edit Data Service (XML Edit)]

4. Navigate to the Queries page and click Add New Query.
5. Since the output is a RDF result set, select output type as RDF from the drop-down list in the Output section.

RDF Base URI has the format rdf:about, which uniquely identifies each resource. In this example, the RDF base URI takes the value of column 1 of the CSV for each row and replaces it with the RDF about attribute inside rdf:Description element.

   RDF Base URI : http://www.product/cd/{id}
   Row namespace : http://www.product/cd#
6. To generate the response in RDF format, click **Add New Output Mapping**. There are two mapping types in RDF Output mapping.

   - as an element
   - as a resource

When mapping an element as a resource, give the resource URI along with the column name that needs to be mapped in curly brackets as shown below. This way, you can link two RDF resources together and create a rela
tionship.

7. Save your changes and click **Next** to add resources. To create the RDF resource, map your RDF query as a resource in the data service.

8. Provide a suitable name and description to your resource. In this example, the resource method is *Get*. Select the the rdfQuery created in the previous step as the Query ID and **Finish**.
9. You can access this RDF resource via a rest call and validate generated RDF response using an online validator such as: http://www.w3.org/RDF/Validator.

Sample Queries

This section gives examples of writing queries using the features discussed so far.

- Sample 1: Using input mappings in a query
- Sample 2: Adding output mappings to a query
- Sample 3: Using a result row namespace
- Sample 4: Querying a RDF data source
- Sample 5: Querying a Web data source
- Sample 6: Querying a google spreadsheet

Sample 1: Using input mappings in a query

This sample demonstrates how to write a simple query with input mappings.

Query ID : customerAddressSQL

SQL Statement : select * from Customers where contactLastName = ? and contactFirstName = ?

The following query needs two parameters for execution. The Input Mapping section is used to specify these input parameters.
For information on adding validations to your input mappings, see Input Validators.

**Sample 2: Adding output mappings to a query**

Following sample shows how to query the Cassandra data source created in Cassandra and add output mappings to it.

**Query ID**: getUsers

**Data Source**: CassandraDatasource

**CQL**: select 'key', 'username', 'password' from USER
You can define how the output of this query looks by adding output mappings as follows:

**Sample 3: Using a result row namespace**

This example shows how to use a results row namespace in output mappings. See Defining Namespaces for information on namespaces.
Sample 4: Querying a RDF data source

Following sample shows a query written for the RDF data source created in Exposing Relational Data in RDF Format, which is based on the added RDF file. This sample uses the following SPARQL query to extract movies information from the data source.

```sparql
PREFIX cd: <http://www.popular.movies/cd#>
WHERE {
}
```

The input mapping section is used to specify parameters to the query. The above query extracts movie information according to the genre. Therefore, we add genre as an input parameter. Next, the output mappings are used to map the response to an output XML.
Sample 5: Querying a Web data source

Following sample shows a query written for the Web data source created in Web Resource. When you add a query to a Web data source, you must enter a Scraper Variable. This scraper variable must be the same as the output name in the Web data source configuration, which returns the output from the configuration. In this example, the var-def name in the configuration is weatherInfo (<var-def name='weatherInfo'/>). Therefore, the Web resource output variable name is also specified as weatherInfo as shown below:
It defines output mappings as follows to specify how the output looks like.
Sample 6: Querying a google spreadsheet

Given below are sample queries that can be written for a google spreadsheet datasource. Note that SQL queries of this type can only be used for a google spreadsheet if the **Query mode** is enabled for the datasource in the data service.

Sample 1:

```sql
SELECT customerNumber, customerName, phone, state, country
FROM customers
```

```xml
<query id="getCustomersSQL" useConfig="gspreadDriverConfig">
<sql>SELECT customerNumber, customerName, phone, state, country FROM customers</sql>
<result element="Customers" rowName="Customer">
  <element column="customerNumber" name="customerNumber" xsdType="string"/>
  <element column="customerName" name="customerName" xsdType="string"/>
  <element column="phone" name="phone" xsdType="string"/>
  <element column="state" name="state" xsdType="string"/>
  <element column="country" name="country" xsdType="string"/>
</result>
</query>
```

Sample 2:
INSERT INTO customers (customerNumber, customerName, contactLastName) VALUES(?,?,?)

```xml
<query id="addCustomerSQL" useConfig="gspreadDriverConfig">
<sql>INSERT INTO customers(customerNumber, customerName, contactLastName, contactFirstName, phone, state, 
<param name="customerNumber" sqlType="STRING"/>
<param name="customerName" sqlType="STRING"/>
<param name="contactLastName" sqlType="STRING"/>
<param name="contactFirstName" sqlType="STRING"/>
<param name="phone" sqlType="STRING"/>
<param name="state" sqlType="STRING"/>
<param name="addressLine1" sqlType="STRING"/>
<param name="addressLine2" sqlType="STRING"/>
<param name="city" sqlType="STRING"/>
<param name="state" sqlType="STRING"/>
<param name="postalCode" sqlType="STRING"/>
<param name="country" sqlType="STRING"/>
<param name="salesRepEmployeeNumber" sqlType="STRING"/>
<param name="creditLimit" sqlType="STRING"/>
</sql>
</query>
```

Sample 3:

```sql
UPDATE customers
SET contactFirstName=?, contactLastName=?
WHERE customerNumber=?
```

```xml
<query id="updateCustomerSQL" useConfig="gspreadDriverConfig">
<sql>UPDATE customers SET contactFirstName=?, contactLastName=? WHERE customerNumber=?</sql>
<param name="contactFirstName" sqlType="STRING"/>
<param name="contactLastName" sqlType="STRING"/>
<param name="customerNumber" sqlType="STRING"/>
</query>
```

Sample 4:

```sql
DELETE FROM customers
WHERE customerNumber=?
```

```xml
<query id="deleteCustomerSQL" useConfig="gspreadDriverConfig">
<sql>DELETE FROM customers WHERE customerNumber=?</sql>
<param name="customerNumber" sqlType="STRING"/>
</query>
```

You can also create new sheets in the Excel or drop existing sheets.

Sample 5:

```sql
CREATE SHEET ProductCategories (ProductCode, Category)
```
Working with DSS Tooling

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

- Installing the DSS Tooling Plug-In
- Creating Data Services Server Artifacts

Installing the DSS Tooling Plug-In

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

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

There are 3 ways to install the tooling plug-in.

- Install WSO2 DSS tooling with pre-packaged Eclipse - This method uses a complete plug-in installation with pre-packaged Eclipse, so that you do not have to install Eclipse separately.
- Install the plug-in on Eclipse Mars using the P2 URL
- Install the plug-in on Eclipse Mars using the P2.zip file

The Install the plug-in on Eclipse Mars using the P2 URL and Install the plug-in on Eclipse Mars using the P2.zip file methods require you to install Eclipse Mars separately in your system, if you do not have it already

Install WSO2 DSS tooling with pre-packaged Eclipse

Download the distribution based on your operating system from here, then extract and run the distribution.

If you are not able to start Eclipse on a Mac, change the security settings in Mac as follows:

1. Go to System Preferences, click Security & Privacy, and then click the General tab.
2. Under Allow apps downloaded from, click Anywhere.

Thereafter, you should be able to start Eclipse by double-clicking the Eclipse icon.
Install the plug-in on Eclipse Mars using the P2 URL

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

4. Give DSS Tool as the name and http://product-dist.wso2.com/p2/developer-studio-kernel/4.1.0/dss-tools/releases/3.5.1 as the location and click OK.
5. Select all the check boxes and click **Next**.

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

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

8. Restart Eclipse to complete the installation.

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

![Add New Software](image)

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

![Add Repository](image)

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

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

10. Restart Eclipse to complete the installation.

Creating Data Services Server Artifacts
The following sections describe how to use WSO2 Developer Studio to create artifacts for Data Services Server. For more information on the details of configuring data service artifacts, including namespaces, service groups, and data source types, see the Data Service Server documentation.

- Creating a data service project
- Creating a data service
  - Creating a new data service
  - Importing a data service
- Enabling security for a data service
- Adding multiple data services
- Creating a custom validator
  - Creating a new custom validator
  - Importing a validator project
- Packaging artifacts

The following section describes how to use WSO2 Developer Studio to add multiple data services to a single Data Service project.

Creating a data service project

1. You can either use the link on the dashboard (click Developer Studio -> Open Dashboard) as shown below or use the File -> New -> Other -> Data Service Project option.

2. On the window that appears, specify a project name.
3. If you want to customize the Maven options (such as including parent POM information in the file from another project in this workspace), click **Next** and specify the options.

4. Click **Finish**.

**Creating a data service**

A data service provides a Web service interface to access data that is stored in relational databases, CSV files, Microsoft Excel sheets, Google spreadsheets, and more. You can create a new data service or import an existing data service from a data service descriptor (DBS) file.

**Creating a new data service**

Follow these steps to create a new data service. Alternatively, you can import an existing data service.

1. You can either use the Data Service Project link on the dashboard (click **Developer Studio -> Open Dashboard**) or right-click on the Data Service project and click **New -> Data Service**.
2. Select **Create New Data Service** and click **Next**.
3. Type unique names for the project and data service.
4. Optionally, specify the service group, namespace, description, location, and working set for this data service.
5. Click **Next** and specify a unique ID for the data source.
6. Specify the type of data source you are creating, and then configure the properties that appear for that type. For details on configuring various data source types, see Creating Using Various Data Sources in the Data Services Server documentation.
7. A Maven POM file will be generated automatically for this project. If you want to customize the Maven options (such as including parent POM information in the file from another project in this workspace), click **Next** and specify the options.
8. Click **Finish**. The data service is created, and the data service is open in the editor. You can now right-click the data service in the outline and add queries, operations, additional data sources, and so on, or click the Source view to add components using XML.
Importing a data service

Follow these steps to import an existing data service from a data service descriptor (DBS) file. Alternatively, you can create a new data service.

1. You can either use the Data Service link on the dashboard (click Developer Studio -> Open Dashboard) or right-click on the Data Service project and click New -> Data Service.
2. Select Import Data Service and click Next.
3. Specify the data service descriptor (DBS) file by typing its full pathname or clicking Browse and navigating to the file.
4. Optionally specify the location and working sets for this project.
5. A Maven POM file will be generated automatically for this project. If you want to customize the Maven options (such as including parent POM information in the file from another project in this workspace), click Next and specify the options.
6. Click Finish. The data service is created, and the data service is open in the editor. You can now right-click the data service in the outline and add queries, operations, additional data sources, and so on, or click the Source view to add components using XML.

Enabling security for a data service

WSO2 supports WS-Security, WS-Policy and WS-Security Policy specifications and these specifications define a behavioral model for Web services. You can find the list of security policies supported by WSO2 DSS from here. These policies are stored in the local registry of your data services server.

To locate the security policies stored in the product registry:

1. Log in to the product's management console and click Registry -> Browse in the Main tab.
2. Navigate to the _system/config/repository/components/org.wso2.carbon.security.mgt/policy folder shown below. See that the available security policies are listed as security scenarios.
3. Click on the required security policy scenario and you will see the registry path to that policy as shown below.
To enable a security policy for a data service:

1. Open your data service in Developer Studio as explained in the Creating a new data service section given above. You can either create a new data service or import an existing DBS file.
2. Add the registry path to the required security policy as shown below.

3. Save the changes.

**Adding multiple data services**

It is possible to add multiple data services to a Data Service project by following the steps described above. The project structure will be as follows:

```
DSS_Project
  \--- dataservice
      \--- DSS_1.dbs
      \--- DSS_2.dbs
      \--- pom.xml
```

**Creating a custom validator**
An **input validator** allows a data service to validate the input parameters in a request and stop the execution of the request if the input doesn’t meet the required criteria. In addition to the default validators provided by Data Service Server, you can create your own custom validators by creating a Java class that implements the `org.wso2.carbon.dataservices.core.validation.Validator` interface. You can **create a new custom validator** or import an **existing validator project**.

**Creating a new custom validator**

Follow these steps to create a new custom validator. Alternatively, you can **import an existing validator project**.

1. You can either use the **Data Services Validator Project** link on the dashboard (click Developer Studio -> Open Dashboard) or click File -> New -> Other -> Data Services Validator Project option.
2. Select **Create New Data Services Validator Project** and click Next.
3. Type a unique name for the project and specify the package and class name for this validator.
4. Optionally, specify the location and working set for this project.
5. A Maven POM file will be generated automatically for this project. If you want to customize the Maven options (such as including parent POM information in the file from another project in this workspace), click Next and specify the options.
6. Click Finish. The project is created, and the new validator class is open in the editor, where you can add your validation logic.

**Importing a validator project**

Follow these steps to import an existing custom validator project. Alternatively, you can create a new custom validator.

1. You can either use the **Data Services Validator Project** link on the dashboard (click Developer Studio -> Open Dashboard) or click File -> New -> Other -> Data Services Validator Project option.
2. Select **Import Project From Workspace** and click Next.
3. Select the existing validator project, and optionally specify the location and working sets for the new project.
4. A Maven POM file will be generated automatically for this project. If you want to customize the Maven options (such as including parent POM information in the file from another project in this workspace), click Next and specify the options.
5. Click Finish. The project is imported, and the validator class is open in the editor, where you can modify the validation logic as needed.

**Packaging artifacts**

You can use a Composite Application project to pack the created data services and then export it as a `.car` file, which can be deployed to the Data Services Server. For information on deploying the artifacts, see **Packaging Artifacts Into Deployable Archives**.
Samples

This section of the documentation will demonstrate how you can try out some sample use cases of WSO2 DSS.

- Samples Setup
- Data Services Clients
- RDBMS Sample
- CSV Sample
- Excel Sample
- Google Spreadsheet Sample
- Nested Query Sample
- Batch Processing Sample
- Eventing Sample
- Secure Data Service Sample
- File App Service Sample
- XSLT Transformation Sample
- Distributed Transactions Sample
- Web Resource Sample
- REST-Style Resources Sample
- JSON Mapping Sample
- Cassandra Sample
- MongoDB Sample
- Custom Datasource Sample
- RDF Sample
- Expose data as RDF
- OData Sample

Samples Setup

To get started on the DSS samples, set up the following in your product server:

- Step 1: Deploy the sample data services
- Step 2: Start the server

Step 1: Deploy the sample data services

In order to run a particular sample, the configuration files (.dbs) relevant to that data service should be deployed in the server. Also, if a data service contains a services.xml file, it should also be deployed along with the .dbs file. Follow the steps given below to deploy the .dbs files and the services.xml files (if applicable) of all the samples that are in the <DSS_HOME>/Samples/ directory.

1. Stop any DSS instances that are running. This is recommended in order to clean the current sample database and to redeploy the services.
2. To clean the current sample database by removing the existing sample data services:
   1. Open the command prompt and go to the <DSS_HOME>/samples directory.
   2. Execute the ant clean command.

   ```
   cd <DSS_HOME>/samples
   ant clean
   ```
3. To redeploy all samples:
   1. Go to the <DSS_HOME>/samples directory in the command prompt.
2. Execute the ant command.

```
<DSS_HOME>/samples
ant
```

Step 2: Start the server

If you have deployed the samples, start the server as explained below.

1. Go to the `<DSS_HOME>/bin` directory in the command prompt.
2. Start the server by executing the command relevant to the OS as shown below.

For Windows OS:

```
cd <DSS_HOME>/bin
./wso2server.bat
```

For Linux OS:

```
cd <DSS_HOME>/bin
./wso2server.sh
```

Data Services Clients

This sample demonstrates how to use code-generated clients with data services. The service stubs are created using WSDL2Java tool.

- Building the sample
- To executing the sample

Building the sample

Deploy the sample data services and start the server using the steps given in Samples Setup.

To executing the sample

1. Open a command prompt and navigate to the `<DSS_HOME>/samples/clients/` directory.
2. Build the clients by executing the ant command as shown below:

```
ant
```

3. You can either run all the generic clients simultaneously or start an individual client.
   - To start all the generic clients (e.g. rdbms, csv, excel, gspread, batch_request), run the ant command as shown below:

```
ant all
```
• To start a specific sample client, run the ant command with the client name. For example, to start the secure_sample client, run the following command:

```
ant secure_sample
```

Listed below are the separate ant commands for each data service client.

- rdbms - RDBMS data source based data service clients.
- csv - CSV data source based data service client.
- excel - Excel data source based data service client.
- gspread - Google Spreadsheet data source based data service client.
- batch_request - Represents a data service client, which executes a batch request in a data service.
- secure_sample - Represents a data service client where a security enabled data service is called. The data service security is enabled manually by the usage of a "services.xml" file in conjunction with the data service.
- file_service_app - Represents a standalone Java application that uses REST style invocations to transfer binary data. This also demonstrates the usage of streaming data, where a file of any size can be uploaded or downloaded using the application.

RDBMS Sample

Relational databases are the most commonly used data sources in data services. The RDBMSample is used here to demonstrate how we can use an RDBMS datasource with WSO2 DSS.

- About the sample data service
- Building the sample
- Executing the sample

About the sample data service

This sample refers to information from a small company, embedded in an H2 database. It contains information such as employee and customer records and product details. The RDBMSample contains separate queries/operations written for specific tasks of the company.

The operations implemented in the service are listed below:

- customersInBoston - Returns all the customers in Boston. This operation is linked to a query, which has the following simple SQL statement, "select * from Customers where city = 'Boston' and country = 'USA'".
- employeesByNumber - Returns employee details, given the employee number. This is an example of a data service operation which takes in a parameter. The linked query's SQL statement is written in the following way, "select * from Employees where employeeNumber = ?", where the "?" represents the parameter which is defined in the query.
- customerAddress - Returns the customer's address, given the person's name.
- productsInfo - This operation lists information on all the products that are produced in the company.
- setEmployeeSalary - Sets the salary of an employee, given the employee number and the revised salary.
- setSalaryForEmployees - Sets the given salary to a given set of employees. Here, the usage of array data type is demonstrated where the employee numbers is an INTEGER ARRAY type.
- incrementEmployeeSalary - Increments the given employee's salary with the given amount.
- addEmployee - Adds a new employee records with the given information.

Building the sample
Deploy the sample data service named RDBMSSample using the steps in Samples Setup.

Executing the sample

You can run the sample service using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

See the topics below for samples on a few important RDBMS functionalities:

- Boxcarring Sample
- Request Box Sample
- Validators Sample
- Default Values Sample
- Query Result Export Sample

Boxcarring Sample

The boxcarring functionality is demonstrated here using the RDBMS Sample. Boxcarring is enabled for this data service and the following control operations are created: "begin_boxcar", "end_boxcar" and "abort_boxcar". Boxcarring is roughly analogous to transactions we see in database systems. Read about boxcarring in Key Concepts and Features in DSS.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Ensure that RDBMSSample is deployed as described in Samples Setup.
2. Log in to the management console of your server and click List under Services in the navigator. See that RDBMSSample is listed here.
3. Click Try this service to open the TryIt tool.
4. Run employeesByNumber operation with the employeeNumber 1002. Output is as follows:

   ![Response](image)

5. Run the "begin_boxcar" operation and the service will switch to a boxcarring session.
6. Run the "incrementEmployeeSalary" operation and give the values "1000" for increment and "1002" for the employeeNumber fields.
7. Open up a different type of a browser for the service to have a fresh session, so it won’t be in the same boxcarring session as earlier. After switching browsers, run the "employeesByNumber" operation with employeeNumber=1002 to check the employee information.
8. Note that the salary is not incremented. The reason is that the request made earlier in boxcarring mode is not executed immediately. It is executed when the boxcarring session ends.
9. Go back to the original browser, run the "incrementEmployeeSalary" operation and give the values "500" for increment and "1002" for the employeeNumber fields. Then, run the "end_boxcar" operation.
10. Next, in any of the two browsers, run the "employeesByNumber" operation with employeeNumber=1002 to check the information. The output will be similar to the following:
As we can see, both "incrementEmployeeSalary" operations are executed at once when the boxcarring session ends with "end_boxcar", resulting in the salary value to 3500. If there's any error in any of the operations inside a boxcarring session, none of the operations will be executed and the service calls will roll-back. A boxcarring session can also be explicitly canceled by calling the "abort_boxcar" operation.

**Request Box Sample**

The Request Box functionality is demonstrated here using the RDBMS Sample Service. This feature is enabled for this data service and the "request_box" operation is created. Read about request box in Key Concepts and Features in DSS.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Ensure that RDBMSSample is deployed as described in Samples Setup.
2. Log in to the management console of your server and click List under Services in the navigator. See that RDBMSSample is listed here.
3. Click Try this service to open the TryIt tool.
4. Select request_box from the list of operations. You are now able to execute multiple operations simultaneously using the request_box operation. Enter values for the operation that you want to execute and remove the other operations from the request_box operation.
5. Click send to execute the request_box operation. You can now see that all the operations that were included in the request_box operation has been executed.

**Validators Sample**

In this sample demonstration, we will use the RDBMS Sample to show how we can validate the input parameters given in a request sent to the service. The "addEmployeeQuery" operation of this service has a length validator for the "lastName" field, where the length of the value should be between 3 and 20. Also, the "email" field is validated using a pattern validator that uses a regular expression to check if it is a real email address.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Ensure that RDBMSSample is deployed as described in Samples Setup.
2. Log in to the management console of your server and click List under Services in the navigator. The RDBMSSample will be listed here.
3. Click Try this service to open the TryIt tool.
4. Select the "addEmployee" operation and enter the following parameter values to the request:
   - employeeNumber: 6001
   - lastName: AB
   - firstName: Nick
   - email: test@test.com
   - salary: 1500
5. Click Send to execute the operation. Note that you are not getting any results. A validation error will show as the response because the "addEmployee" operation has failed. This is because the lastName only has 2 characters.
6. Now, change the lastName and the email address in the request as shown below and execute the operation again.
   - employeeNumber: 6001
6. Note that the operation will fail again because the email address is invalid.

7. Next, execute the "addEmployee" operation again with parameter values for email address and lastName as shown below:
   - employeeNumber: 6001
   - lastName: ABC
   - firstName: Nick
   - email: test@test.com
   - salary: 1500.

Note that the entry will be successfully added and you will get a result.

**Default Values Sample**

This example demonstrates how default values specified in the .dbs file of the data service will be used when specific values are not given for some fields. In the RDBMS Sample service, a default employee salary is specified as 1500 in the .dbs file as shown below.

```xml
<param defaultValue="1500" name="salary" ordinal="5" paramType="SCALAR" sqlType="DOUBLE" type="IN"/>
```

Therefore, when you use the "addEmployee" operation to add a new employee, if you do not specify an explicit salary value for the employee, the default salary will be added. For example, execute the "addEmployee" operation with the following parameter settings:

- employeeNumber=6002;
- lastName=Smith;
- firstName=John;
- email=john@smith.com
- Remove <xs:salary xmlns:xs="http://ws.wso2.org/dataservice/samples/rdbms_sample" />

Now, when you execute the "addEmployee" operation, the result will be the following:
Note that the "salary" value is set to 1500. This is the default value for that field as specified in the .dbs file of the data service.

Query Result Export Sample

The operations "thousandFive" and "incrementEmployeeSalaryEx" are used in demonstrating the query result export functionality in a data service. The query used in "thousandFive" operation exports the a result value with the name "increment". This increment value is taken in as a parameter by the query in the "incrementEmployeeSalaryEx" operation.

The figure below shows the dbs fragment that defines this behaviour:

```xml
<query id="thousandFiveQuery" useConfig="default">
  <sql>select 1500 as value</sql>
  <result element="Numbers" rowName="Number">
    <element name="value" column="value" export="increment" xsdType="xs:integer" />
  </result>
</query>
<query id="incrementEmployeeSalaryQuery" useConfig="default">
  <sql>update Employees set salary=salary? where employeeNumber=?</sql>
  <property name="org.wso2.ws.dataservice.autocommit">true</property>
  <property>
    <param name="increment" sqlType="DOUBLE" ordinal="1" />  
  </property>
  <param name="employeeNumber" sqlType="INTEGER" ordinal="2" />
</query>
```

- employeeByNumber(1002)
- begin_boxcar()
- thousandFive()
- incrementEmployeeSalaryEx(1002)
- end_boxcar()
- employeeByNumber(1002)

The return values from the first "employeeByNumber()" call and the second have a difference of 1500, which is the amount incremented within the boxcarring session. The 1500 value is passed into "incrementEmployeeSalaryEx" operation by the "thousandFive" operation.

CSV Sample

Using CSV files is an easy and common way of storing simple tabular data. This sample demonstrates the usage of a CSV file as a data source for data services in WSO2 DSS.

- About the sample
- Building the sample
- Executing the sample

About the sample

The CSV file (Products.csv), stored in the <DSS_HOME>/samples/resources/ folder contains data about the
products that are manufactured in an automobile company. Products refer to cars/motorcycles. The data table has the following columns: "ID", "Name", "Classification" and "Price". The sample data service used in this demonstration (CSVSampleService.dbs) contains a single operation named getProducts, which lists all the products that are manufactured in the company.

See About Data Services and Resources for a definition of data services and operations.

Building the sample

The sample data service named CSVSampleService should be deployed using the instructions in Samples Setup.

Executing the sample

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click List under Services in the navigator. The CSVSampleService will be listed here.
2. Click Try this service to open the TryIt tool.
3. Select the "getProducts" operation and click Send to execute the request. You will get a product list in the response as shown below.

Excel Sample

Microsoft Excel is one of the most popular spreadsheet formats. An Excel sheet can be used as a data source for creating data services in WSO2 Data Services Server.

- About the sample
- Building the sample
- Running the sample

About the sample

The Products.xls file, stored in the <DSS_HOME>/samples/resources/ folder is used as the datasource to demonstrate this sample data service. This datasource contains details of the products manufactured in an automobile company. Products refer to cars/motorcycles. The data table has the following columns: "ID", "Name", "Classification" and "Price". The sample data service used in this demonstration (CSVSampleService.dbs) contains a single operation named getProducts, which lists all the products that are manufactured in the company.

See About Data Services and Resources for a definition of data services and operations.

Building the sample

The sample data service named CSVSampleService should be deployed using the instructions in Samples Setup.

Executing the sample

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click List under Services in the navigator. The CSVSampleService will be listed here.
2. Click Try this service to open the TryIt tool.
3. Select the "getProducts" operation and click Send to execute the request. You will get a product list in the response as shown below.
"Classification" and "Price" (same as in the CSV Sample). This sample data service contains two operations:

- **getProducts** - Lists all the products that are manufactured in the company. Each result record will contain "ID" and "Name" fields.
- **getProductClassifications** - Lists all the products along with the classifications. The result is presented with two records: "Name" and "classification".

See About Data Services and Resources for a definition of data services and operations.

**Building the sample**

The sample data service named ExcelSampleService should be deployed using the instructions in Samples Setup.

**Running the sample**

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click **List** under **Services** in the navigator. The ExcelSampleService will be listed here.
2. Click **Try this service** to open the TryIt tool.
3. Select the "getProducts" operation and click **Send** to execute the operation. You will get a product list in the response as shown below.

![Image of product list response]

**Google Spreadsheet Sample**

A data sharing format used often in the Internet is Google Spreadsheet. The WSO2 Data Services Server contains support for Google Spreadsheets as a data source.

- About the sample
- Building the sample
- Running the sample

**About the sample**

The spreadsheet contains data about the customers of a specific company. The GSpread sample service contains a
single operation named *getCustomers*, which returns all the customer information that is stored in the Google spreadsheet. There is also a resource defined as *customers*, which can be accessed using the following URL: "http://localhost:9763/services/GSpreadSample.HTTPEndpoint/customers".

See About Data Services and Resources for a definition of data services and operations.

Building the sample

The sample data service named *GSpreadSample* should be deployed and configured as follows:

1. Follow the instructions in Samples Setup to have the sample deployed.
2. Because we are using a Google spreadsheet as the datasource, we need to update the credentials for accessing the spreadsheet. This is done using a **Client ID**, **Client Secret** and **Refresh Token** that is generated from the **Google API Console**. Follow the google documentation for instructions on how to get an OAuth2 client ID and client secret. In order to get these credentials, you will be asked to provide an **authorization redirect URL**. Be sure to use localhost as the IP in the **browser redirect URL**.
3. Now you can add the credentials to the GSpreadSample data service as follows:
   1. Log in to the management console of your server using the following URL: https://localhost:9443/carbon/
   2. Click **List** under Services in the navigator. The **GSpreadSample** will be listed there.
   3. Click the **GSpreadSample** to open the sample's dashboard.
   4. Click **Edit Data Service Wizard** to start editing the data service using the management console.
   5. Click **Next** until you reach the **Datasource** screen shown below.
   6. Click the **GSpreadSample** datasource to start editing the datasource.
   7. Note that a client ID, client secret and refresh token are specified by default. Replace the **Client ID** and **Client Secret** values using the new values you obtained in step 2 above.
   8. Click **Generate Token**. You will be redirected to the google consent page. After you approve that, the refresh token will be inserted into the **Refresh Token** field as shown below.

   For the purpose of this sample, be sure to use localhost as the IP in the above URL.

9. Save the datasource. You can now run the sample as explained below.
Running the sample

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. In the management console of your server and click List under Services in the navigator. The GSpreadSamp le will be listed here.
2. Click Try this service to open the TryIt tool.
3. Select the "getCustomers" operation and click Send to execute the request. You will get the customer details in the response as shown below.

```xml
<Customers xmlns="http://ws.wso2.org/dataservice/samples/gspread_sample_service">
  <Customer customerName="Atelier graphique" city="Nantes">
    <customerNumber>103</customerNumber>
  </Customer>
  <Customer customerName="Signal Gift Stores" city="Las Vegas">
    <customerNumber>112</customerNumber>
  </Customer>
  <Customer customerName="Australian Collectors, Co." city="Melbourne">
    <customerNumber>114</customerNumber>
  </Customer>
  <Customer customerName="La Rochelle Gifts" city="Nantes">
    <customerNumber>119</customerNumber>
  </Customer>
  <Customer customerName="Baane Mini Imports" city="Stavern">
    <customerNumber>121</customerNumber>
  </Customer>
</Customers>
```

Nested Query Sample

In a data service, sometimes the result of one query may be used as an input parameter for another query. In such situations, we use the nested query feature available in the WSO2 Data Services Server.

- About the sample
- Building the sample
- Running the sample

About the sample

The sample data service (NestedQuerySample) contains the following operations:

- employeesInOffice - The employeesInOfficeSQL query takes in an office code and prints the details of all the employees in that office.
- listOffices - The listOfficesSQL query lists all the offices in the database. When office information is printed, all the employees that belong to an office are also printed using a nested query, which refers to the employee sInOfficeSQL query, which is invoked by the "employeesInOffice" operation.
- customerNameSQL - The customerNameSQL query takes in the customer number as a parameter and returns the customer name. In the result section of this query, there is no "rowName" defined (only an element wrapper). With this option, a separate element will not be created for every record in the result. However, the record content will be directly added to the outer element. This is best used in a query where there is only one record to be returned.
- customerOrders - The customerOrdersSQL query prints the details of all the orders that are in the database. In the result elements, a nested query which references the customerNameSQL query is used. In this way, the name of the customer who placed the order can also be returned with the order details.
See About Data Services and Resources for a definition of data services and operations.

Building the sample

The sample data service NestedQuerySample should be deployed using the instructions in Samples Setup.

Running the sample

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click List under Services in the navigator. The Nested QuerySample will be listed here.
2. Click Try this service to open the TryIt tool.
3. Select an operation and click Send to execute the operation. You will get the product list in the response as shown below.

- Operation invoking the listOfficesSQL query:

![List Offices Response]

- Operation invoking the customerOrdersSQL query:

![Customer Orders Response]

- Operation invoking the customerNameSQL query:

![Customer Name Response]
Enter "103" as the customer name and send the request as shown below.

- Operation invoking the employeesInOfficeSQL query:

First, see the result of "listOffices" operation to get the office code. Then, enter the office code ("1") and send the request as shown below.

Batch Processing Sample

In this sample demonstration, we use a data service with batch processing. Batch processing is used when a specific in-only operation must be executed sequentially with different sets of parameters.

- Building the sample
- Running the sample

Building the sample

The sample data service named BatchRequestSample should be deployed using the instructions in Samples Setup.

Running the sample

The command line application is used here to present the functionality of batch processing. As shown in the Data Services Clients section, execute the “ant batch_request” command to run the sample. The output will resemble the
The sample tests batch processing by executing a batch operation with three parameter tests, which means, it effectively executes three consecutive operations. The operation is to add a new employee to the database. Here we execute the batch operation and checks for the existence of the three newly added entries. A requirement of batch operations is, if one invocation fails, then all the other operations in the batch should fail. This can be seen here, where the sets of three employee entries are either inserted together, or none of them are inserted. In this sample, a validator is used for checking if a valid email address is given when adding a new employee entry to the database. If the email is invalid, that operation invocation fails.

In this sample data service, there are four such batch requests executed as explained below.

1. All the parameters are added properly and all operations executed successfully. Therefore, when checking for entries with ids "180000", "180001", "180002", all returns true.
2. The request with id "180003" is executed successfully. But when it reaches "180004", a validation error occurs since the email address is invalid. All service invocations in that batch request are rolled back. So in the next section when checking for the employee ids belonging to this batch request, no one would have been added to the database. This is expected since all the operations in a batch operation are executed in a single transaction.
3. In this instance, the 3rd operation in the batch request is not given an email address. It causes a validation error. The transaction is rolled back and all ids "180006", "180007" and "180008" are not executed.
4. The requests here are all given valid parameters; same as in the 1st request. So all the records "180009", "180010" and "180011" are added successfully.

This demonstrates how the batch operations behave in a transactional manner.

**Eventing Sample**

**Eventing** is used to send notifications when a particular action is completed, such as when an operation with particular arguments is invoked or when a specific response is returned. Notifications are enabled in WSO2 DSS using input and output event triggers.
About the sample data service

The *EventingSample* data service used here is based on the functionality of managing the inventory of a car/motorcycle dealership. It contains the following queries:

- addProduct: A new product is added to the catalog using the *addProductQuery* query.
- updateProductQuantity: The availability of a certain product is updated using the *updateProductQuantityQuery* query. This is linked with an input event trigger, which monitors whether the stock of this product is critically low. If so, it fires an event notifying the subscribers that this product's stock needs to be updated.
- getProductByCode: When the product code is given, the *getProductByCodeQuery* query retrieves product information. The company gives special attention to people querying the database for motorcycles. So when a query is made where the product type is "motorcycle", an output trigger associated with this query sends messages to the respective subscribers, which is our email address in this sample.

Pre-requisites

1. In the sample, we define a specific topic for events to be published, and an email address is added as a subscription for that topic. In order to allow mails to be sent, we have to configure the mail sender in the client Axis2 configuration. A sample *axis2.xml* file for the client configuration is stored in `<DSS_HOME>/samples/resources/sample_axis2_client.xml`. Use it to replace the `<DSS_HOME>/repository/conf/axis2/axis2_client.xml` file. The *axis2_client.xml* file contains an already configured mail sender section and is ready to be used.

2. Edit the data service by adding a valid email address: Here we use the **XML Edit** option to edit the dataservice. In the XML view, replace all occurrences of "test@test.com" with your own email address and restart the server.

Building the sample

The .dbs file of this sample data service should be manually deployed in your server as explained below.

1. Log in to the management console and select **Upload** under **Data Service** menu.
2. Select the *EventingSample.dbs* file from the `<DSS_HOME>/samples/dbs/rdbms/` folder and click **Upload**.
3. If the file is deployed successfully, the **Deployed Services** window appears with the new data service listed. Alternatively, copy the file to the `<DSS_HOME>/repository/deployment/server/dataservices` folder. It will be deployed instantly as hot deployment, which is enabled in Data Services Server by default.

See [Uploading a Created Data Service](#).

Running the Sample

The sample service can be run using the **TryIt** tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the **Data Services Clients** section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click **List** under **Services** in the navigator. The *EventingSample* will be listed here.
2. Click **Try this service** to open the TryIt tool.
3. Select the relevant operation and click **Send** to execute the command as shown below.

```text
- We will first add a new product into the inventory by executing the "addProduct" operation:
```
After adding the product, we will update the availability of it by changing the product quantity in stock. Execute the "updateProductQuantity" operation:

```xml
<addProduct xmlns:p="http://ws.wso2.org/dataservice/samples/eventing_sample">
  <productCode>s20_1001</productCode>
  <productLine>Classic Cars</productLine>
  <productName>Pontiac Firebird</productName>
  <quantityInStock>30</quantityInStock>
  <buyPrice>4550.00</buyPrice>
</addProduct>
```

Here we are setting the "quantityInStock" to 5. Because of the event trigger, when the stock is less than 10, an event is fired. Soon you will receive an event notification to your mail inbox. Figure below shows the
contents of the email attachment.

```
- <soapenv:Envelope>
  - <soapenv:Header>
    <ns:topic>product_stock_low_topic</ns:topic>
  </soapenv:Header>
- <soapenv:Body>
  - <data-services-event>
    <service-name>EventingSample</service-name>
    <query-id>updateProductQuantityQuery</query-id>
    <time>Fri May 07 07:34:41 IST 2010</time>
    - <content>
      - <updateProductQuantityQuery>
        <quantityInStock>5</quantityInStock>
        <productCode>S20_1001</productCode>
      </updateProductQuantityQuery>
      - <updateProductQuantityQuery>
        <quantityInStock>5</quantityInStock>
        <productCode>S10_1678</productCode>
      </updateProductQuantityQuery>
    </content>
  </data-services-event>
</soapenv:Body>
</soapenv:Envelope>
```

- Here we are querying the inventory system for a specific motorcycle model by executing the "getProductByCode" operation.

By making a request with the motorcycle product line, we are triggering an event associated with it. An email will be sent regarding this query. The contents of the email is as follows:

```
- <soapenv:Envelope>
  - <soapenv:Header>
    <ns:topic>product_line_motorcycles_topic</ns:topic>
  </soapenv:Header>
- <soapenv:Body>
  - <data-services-event>
    <service-name>EventingSample</service-name>
    <query-id>getProductByCodeQuery</query-id>
    <time>Fri May 07 07:33:32 IST 2010</time>
    - <content>
      - <Products>
        - <Product>
          <productCode>S10_1678</productCode>
          <productLine>Motorcycles</productLine>
          <productName>1969 Harley Davidson Ultimate Chopper</productName>
          <quantityInStock>7933</quantityInStock>
        </Product>
      </Products>
    </content>
  </data-services-event>
</soapenv:Body>
</soapenv:Envelope>
```

Secure Data Service Sample
You may want to secure a data service by authentication, encryption and the usage of signatures. This is facilitated in the WSO2 Data Services Server by applying security properties directly to the data service. This sample demonstrates how a service client is used to access a secured data service.

- About the sample
- Building the sample
- Securing the Data Service
- Running the sample

About the sample

This service contains a single query/operation named `showAllOffices`, which returns all the office branches in a company.

Secure service client: When using a service client to access a secured data service, it must follow special steps in creating a secured connection to the service. The following code snippet is taken from the Axis2 service client used to access our sample secure data service.

```java
String epr = "https://" + HOST_IP + ":" + HOST_HTTPS_PORT + "/services/SecureDataService";
System.setProperty("javax.net.ssl.trustStore", (new File(CLIENT_JKS_PATH)).getAbsolutePath());
ConfigurationContext ctx = ConfigurationContextFactory.createConfigurationContextFromFileSystem(null, null);
SecureDataServiceStub stub = new SecureDataServiceStub(ctx, epr);
ServiceClient client = stub._getServiceClient();
Options options = client.getOptions();
client.engageModule("rampart");
options.setUserName("admin");
options.setPassword("admin");
options.setProperty(RampartMessageData.KEY_RAMPART_POLICY, loadPolicy(SECURITY_POLICY_PATH));
```

First, the client key store file path (CLIENT_JKS_PATH) is set as a Java system property. The next few lines are Axis2 specific code to initiate the Axis2 runtime and its security module, Rampart. You can see by the end the security policy path (SECURITY_POLICY_PATH) is given to be processed by Rampart. In the security policy, the runtime is notified that we are securing the service and using UsernameToken as the authentication method. After these steps are successfully carried out, we can use the service client to make secure service calls to our data service.

Building the sample

The sample data service, SecureDataService should be deployed using the instructions in Samples Setup.

Securing the Data Service

Enable security for the data service SecureDataService. Select UsernameToken in the basic scenario. Here we are simply enabling username/password based authentication for the data service. Select everyone as the user group. For step-by-step instructions on service-level security setting, refer to Security for Web Services.

Running the sample
The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

The command line application is used here to present the functionality of the secured data service. As shown in Data Services Clients, run the "ant secure_sample" command to run the sample. The output is as follows:

```
[java] Office Code: 1
[java] Phone: +1 650 219 4782
[java] Address Line 1: 100 Market Street
[java] Address Line 2: Suite 300
[java] City: San Francisco
[java] State: CA
[java] Postal Code: 94080
[java] Country: USA
[-----------------------------
[java] Office Code: 2
[java] Phone: +1 215 837 0825
[java] Address Line 1: 1550 Court Place
[java] Address Line 2: Suite 102
[java] City: Boston
[java] State: MA
[java] Postal Code: 02107
[java] Country: USA
[-----------------------------
[java] Office Code: 3
[java] Phone: +1 212 555 3000
[java] Address Line 1: 523 East 53rd Street
[java] Address Line 2: apt. 5A
```

The above demonstration shows how a Java service client can be used in accessing a secured data service.

**File App Service Sample**

- About the sample
- Building the sample
- Running the sample

**About the sample**

This sample demonstrates a file upload/download/preview application with the use of several features available in the WSO2 Data Services Server. Some notable functions are listed below.

- **Streaming Support**: With the data streaming functionality, when a service client makes a request, the result is streamed to the service client rather than building the full result in the server and returning it. This allows virtually unlimited payload sizes in the result, and the response is instantaneous to the client.
- **Binary Data**: Binary data is handled using Base64 encoding to store and retrieve non-textual data, such as data files and BLOB fields in databases.

**Service description**

The sample data service used here (FileService) transfers and retrieves file data between the client and a database. The list of queries used by the data service to carry out the actions are as follows.

- **createNewFileQuery**: Creates a new file when the file name and type is given.
- **appendDataToFileQuery**: Appends new data to a given file. By sending in file chunks, data streaming can also be done while sending the data.
• deleteFileQuery: Deletes a file when the file name is given.
• checkFileExistsQuery: Checks if a file with a given file name exists.
• getFileRecordsQuery: Returns all data records associated with a specific file.
• getFileNamesQuery: Returns the names of the files in the database.
• getFileTypeQuery: Returns the file type of a given file name.
• getFileSizeQuery: Returns the size of a given file name.

By combining functionality of the above queries, we can create a sample file management system with the use of data services. See About Data Services and Resources for a definition of data services, queries and operations.

Building the sample

The sample data service, FileService should be deployed using the instructions in Samples Setup.

Running the sample

To execute the application, you have to run the ant file_service_app command as described in the Data Services Clients section.

• Uploading files
  When uploading files, after a file is selected to be sent, the file is read and the data is transferred in chunks to the server. Figure below shows a file upload in progress.

• Downloading files
  Select a file you want to download from the file list and press download. After selecting the path to be saved, the file data is retrieved from the server and saved on the local disk. When the user clicks download, the
data download starts immediately as indicated by the progress bar. This is possible because of the data streaming capability of the WSO2 Data Services Server.

- **Previewing files**
  Images can be previewed using this application. Select an image file and click **Preview**. The image is downloaded to memory and shown in the preview panel:

- **Deleting files**
  To delete a file from the server, select the file name from the list and press **Delete**. This invokes a data service operation and removes the file from the database.

**XSLT Transformation Sample**

XSLT transformation is used in WSO2 DSS to transform the result of an already defined operation into a different result. The user can define the transformation xslt and provide the url of the transformation file in the result element.

- **About the sample**
- **Building the sample**
- **Running the sample**

**About the sample**

The sample data service (*ExcelSampleService*), which we use for this demonstration consists of two operations:
• getProducts - Lists all the products that are manufactured in the company. Each record in the result contains "ID" and "Name" fields.
• getProductClassifications - Lists all the products along with the classifications. The result will consist of two records, "Name" and "classification". However, note that because of the xslt transformation, the actual output of this operation has changed to "product-Name" and "product-classification".

See About Data Services and Resources for a definition of data services and operations.

Building the sample

The sample data service named ExcelSampleService should be deployed using the instructions in Samples Setup.

Running the sample

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click List under Services in the navigator. The ExcelSampleService will be listed here.
2. Click Try this service to open the TryIt tool.
3. Select "getProductClassifications" operation and click Send to execute. You will get the response as shown below.

Distributed Transactions Sample

Distributed transactions are important when we coordinate multiple transactions across multiple databases. This sample demonstrates the support for distributed transactions in WSO2 Data Services Server.

• About the sample
• Building the sample
• Running the sample

About the sample
The scenario used in this sample data service (*DTPSampleService*) is based on two banks using separate databases. This data service uses distributed transactions with multiple data sources.

**Building the sample**

The sample data service *DTPSampleService* should be deployed using the instructions in the *Samples Setup* section.

**Running the sample**

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the *Data Services Clients* section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click **List** under **Services** in the navigator. The *DTPSampleService* will be listed here.
2. Click **Try this service** to open the TryIt tool. Execute the following operations:
3. Run the **addAccountToBank1** operation with balance 1000. This creates a new account in bank1 and returns the bank id. Output is as follows:

   ![Request and Response for addAccountToBank1](image)

4. Run the **addAccountToBank2** operation with balance 2000. This creates a new account in bank2 and returns the bank id. Output is as follows:

   ![Request and Response for addAccountToBank2](image)

5. Run the **begin_boxcar** operation. The service will switch to a boxcarring session.
6. Run the **addToAccountBalanceInBank1** operation and give the values "1" for "accountId" field and "-100" for the "value" field.
7. Run the **addToAccountBalanceInBank2** operation and give the values "1" for "accountId" field and "h200" for the "value" field. Here we are giving an incorrect value to the "value" field. So, if distributed transaction works correctly, the whole transaction (including the previous operation done on addToAccountBalanceInBank1) should be aborted even though the transaction involves two different databases.
8. Run the **end_boxcar** operation.
9. We can run the **getAccountBalanceFromBank1** and **getAccountBalanceFromBank2** operations to verify that the transactions have aborted and that distributed transaction works properly.
10. Run the **getAccountBalanceFromBank1** operation. The output is as follows:

    ![Request and Response for getAccountBalanceFromBank1](image)

11. Run the **getAccountBalanceFromBank2** operation. The output is as follows:
As we can see, the `addAccountToBank1` service call also rolled back because of the error in `addAccountToBank2` operation.

**Web Resource Sample**

- About the sample
- Building the sample
- Running the sample

**About the sample**

WSO2 Data Services Server supports scraping and extracting Web information and exposing it as dataservices. This sample demonstrates how to extract book information using Site Point (http://www.sitepoint.com/html). The Web Resource sample service contains a single operation named `getBooksInfo`, which returns a list of books and authors.

**Building the sample**

Deploy the sample data service named `WebResourceSample`, using the instructions in Samples Setup.

**Running the sample**

The sample service can be run using the TryIt tool, which comes with the Data Services Server. You can also run it using a code-generated java client. See Data Services Clients for details.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click List under Services in the navigator. The `WebResourceSample` will be listed here.
2. Click Try this service to open the TryIt tool.
3. Select the "getBooksInfo" operation and click Send to execute the operation. You will get the response as shown below.
REST-Style Resources Sample

This sample demonstrates how data can be exposed as REST resources through a data service and also how the service can be invoked in REST style.

- About the sample
- Building the sample
- Running the sample

About the sample

This sample scenario is based on a database which stores information about the products of an enterprise. There are separate queries/resources written for specific tasks of the products. Resources are defined in order to access data through REST calls. The resources implemented in the service are listed below.

- Create Product - Use HTTP POST request to insert new products.
  Resource Path : product
  Resource Method : POST

- Update Product - Use HTTP PUT request to update product information.
  Resource Path : product
  Resource Method : PUT

- GET Product - Use HTTP GET request to retrieve product information for a given product code.
  Resource Path : product/{productCode}
  Resource Method : GET

- Get All Products - Use HTTP GET request to get all the products.
  Resource Path : products
  Resource Method : GET

- Delete Product - Use HTTP DELETE request to delete a given product from the database.
  Resource Path : product/{productCode}
  Resource Method : DELETE

See About Data Services and Resources for details.
Building the sample

The sample data service ResourcesSample should be deployed using the instructions in Samples Setup.

Running the sample

The service can be invoked in REST-style via curl (http://curl.haxx.se). The following commands can be executed using the command line to invoke each of the resources.

- **Create Product**

  ```bash
curl -X POST -d 
  --header 'Content-Type: application/xml'
  http://localhost:9763/services/samples/ResourcesSample.HTTPEndpoint/product
  ```

- **Update Product**

  ```bash
curl -X PUT -d 
  --header 'Content-Type: application/xml'
  http://localhost:9763/services/samples/ResourcesSample.HTTPEndpoint/product
  ```

- **GET Product**

  ```bash
curl -X GET
  http://localhost:9763/services/samples/ResourcesSample.HTTPEndpoint/product/S10_100
  ```

- **Get All Products**

  ```bash
curl -X GET
  http://localhost:9763/services/samples/ResourcesSample.HTTPEndpoint/products
  ```

- **Delete Product**

  ```bash
curl -X DELETE
  http://localhost:9763/services/samples/ResourcesSample.HTTPEndpoint/product/S10_100
  ```
JSON Mapping Sample

- About the sample
- Building the sample
- Running the sample

About the sample

A popular data format used with RESTful services is JSON. This sample contains a data service with JSON output mapping templates. Go to Exposing Data in JSON Format for details on how to use JSON templates for output mapping in a data service.

The service contains the following resources:

- Resource: "offices"
  - Method: POST
  - Description: Lists all the offices in the database. When office information is printed, all the employees that belong to an office are also printed using a nested query, which refers to the employeesInOfficeSQL query.

- Resource: "boston_customers"
  - Method: GET
  - Description: Returns all the customers in Boston.

- Resource: "employee/{employeeNumber}"  
  - Method: GET
  - Description: Returns employee information for the given employee number.

- Resource: "employee"
  - Method: POST
  - Description: Insert a new employee to the database with the given information.

- Resource: "employee" with batch request
  - Method: POST
  - Description: Insert a set of new employees to the database with the given information.

- Resource: "employee"
  - Method: PUT
  - Description: Updates an existing user with new information.

- Resource: "employee"
  - Method: GET
  - Description: Retrieve an existing employee.

Building the sample

The sample data service named JSONSample should be deployed using the instructions in Samples Setup.

Running the sample

The sample service can be run using any client that can do REST calls. For example, use tools such as cURL or

```
curl -X DELETE
http://localhost:9763/services/samples/ResourcesSample.HTTPEndpoint/product/S10_100
```
Advanced REST Client Chrome browser extension. The HTTP header "Accept" should be set to "application/json" and also, if a JSON payload is sent to the server, the HTTP header "Content-Type" should be set to "application/json".

The HTTP requests sent for each of the resources using cURL would be as follows:

- **"offices" (method=GET)**

  ```
  curl -X GET -H "Accept: application/json"
  http://localhost:9763/services/JSONSample/offices
  ```

- **"boston_customers" (method=GET)**

  ```
  curl -X GET -H "Accept: application/json"
  http://localhost:9763/services/JSONSample/boston_customers
  ```

- **"employee/{employeeNumber}" (method=GET)**

  ```
  curl -X GET -H "Accept: application/json"
  http://localhost:9763/services/JSONSample/employee/1002
  ```

- **"employee" (method=POST) :** A sample payload saved as the employee-payload.json file, that can be used when sending the data is given below.

  ```
  {
  "_postemployee": {
    "employeeNumber": 14001,
    "lastName": "Smith",
    "firstName": "Will",
    "email": "will@google.com",
    "salary": 15500.0
  }
  }
  ```

  The HTTP request sent using CURL would be as follows:

  ```
  curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json' --data "@employee-payload.json"
  http://localhost:9763/services/JSONSample/employee/
  ```

- **"employee" (method=POST) with batch request :** A sample payload saved as the employee-payload.json file, that can be used when sending the data is given below.
The HTTP request sent using CURL would be as follows:

```
curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json' --data @employee-payload.json
http://localhost:9763/services/JSONSample/employee_batch_req
```

- "employee" (method=PUT) - A sample payload saved as the employee-payload-update.json file, that can be used when sending the data is given below.

```
{
   "_putemployee": {
      "employeeNumber" : 14001,
      "lastName": "Smith",
      "firstName": "Will",
      "email": "will@smith.com",
      "salary": 78500.0
   }
}
```

The HTTP request sent using CURL would be as follows:

```
curl -X PUT -H 'Accept: application/json' -H 'Content-Type: application/json' --data @employee-payload-update.json
http://localhost:9763/services/JSONSample/employee/
```
• "employee" (method=GET)

```bash
curl -X GET -H 'Accept: application/json'
http://localhost:9763/services/JSONSample/employee/14001
```

**Cassandra Sample**

This sample demonstrates the use of the Cassandra data source functionality in WSO2 DSS, which you can use to connect to a Cassandra cluster.

- **Prerequisites**
- **Building the sample**
- **Running the sample**

**Prerequisites**

A Cassandra server of version 1.2.x, 2.0 or 3.0 should be already running in the default port. Cassandra version 3.0 is recommended from DSS 3.5.1 onwards.

**Building the sample**

The sample data service *CassandraSample* should be deployed using the instructions in *Samples Setup* section. Go to *Cassandra* for details on how to create Cassandra data sources.

**Running the sample**

The sample can be run using any SOAP client such as the *Tryit tool* that comes bundled with WSO2 DSS.

**Service description**

The service contains the following operations:

- "createKS": This creates the Cassandra keyspace required for the sample. This must be called once before running other operations.
- "createTable": This creates a Cassandra CQL3 table required for the sample. This must be called once after "createKS" is called.
- "addUser": This operation takes in user details and adds it to Cassandra. Here "ID" is the primary key and it takes in an UUID as the value.
- "addUser_batch_req": This is the corresponding batch request of "addUser", where it takes in a batch of "addUser" data entries and adds it as a single batch operation.
- "getUserById": This operation returns the user information when you provide the ID of a specific user.
- "getUsers": This operation returns information of all the users stored in the Cassandra store.

**MongoDB Sample**

MongoDB is a NoSQL type, open-source document database. This sample demonstrates the usage of MongoDB as a data source in WSO2 DSS.

- **About the sample**
- **Prerequisites**
- **Building the sample**
- **Running the sample**
About the sample

This sample data service contains the operations listed below. See About Data Services and Resources for a definition of data services and operations.

- **mongo_insert**: This operation adds a document according to the provided id, name.
- **mongo_insert_doc**: Using this operation you can inserts a document into the 'things' collection.
- **mongo_find**: This operation returns all documents from the collection.
- **mongo_count**: This operation counts and returns the number of all documents in the 'things' collection
- **mongo_update**: This operation sets the name as 'Zack' and the id as the provided value for the document where name is 'Bob'
- **mongo_remove**: This operation removes all the documents from the collection 'things' where id is equal to the given value
- **mongo_drop**: This operation will drop the collection 'things' from the database

Prerequisites

A MongoDB server v2.4.x or v2.2.x should be already running in the default port. Create a collection as below in the command shell.

```bash
mongo
use mydb
db.createCollection("things")
db.things.insert( { id: 1, name: "Document1" } )
```

Building the sample

The sample data service named **MongoDBSample** should be deployed using the instructions in the Samples Setup section.

Running the sample

The sample can be run using any SOAP client such as the **Tryit tool** that comes bundled with WSO2 DSS.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click **List** under Services in the navigator. The **MongoDBSample** will be listed here.
2. Click **Try this service** to open the TryIt tool.
3. Select the relevant operation and click **Send** to execute the commands as shown below.
   - Invoking the operation 'mongo_insert' to insert a document
Invoking the operation 'mongo_find' to retrieve the data in the collection

Custom Datasource Sample

This sample demonstrates the use of custom datasources in a data service, using WSO2 DSS.

- About the data service
- Building the sample
- Running the sample

About the data service

The InMemoryDSSample used here consists of the following custom datasource implementations:


See how data services are created using custom datasources.

Building the sample

The sample data service InMemoryDSSample should be deployed using the instructions in Samples Setup section.

Running the sample

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click List under Services in the navigator.
2. The InMemoryDSSample should be listed.
3. Click Try this service to open the TryIt tool for the InMemoryDSSample.
4. Select the relevant operation and click Send to execute the command. The service contains the following operations:
• "addVehicle": This operation adds a new vehicle to the datasource.

• "getOldVehicles": This gives all the vehicles added with the year greater than 1970.

• "getAllVehicles": This operation returns all the vehicles added in the datasource.

• "getVehicleById": This operation gives the vehicle information which matches with the provided id.
• "getAllUsers": This operation returns the user information stored in the datasource.

• "echo1" and "echo2": This operation returns the entered values.
RDF Sample

RDF is a flexible and extensible way to represent information about web resources. This sample demonstrates the usage of an RDF file as a data source for data services in WSO2 DSS.

About the sample

The RDF file (Movies.rdf), stored in the `<DSS_HOME>/samples/resources/` folder contains data about some popular movies. Each movie data has the following sub elements: "title", "director", "year", "genre" and "actor". The sample data service used in this demonstration (RDFSampleService.dbs) contains two operations named `getAllMovieData` and `getMoviesByGenre`.

See Data Services and Resources for a definition of data services and operations.

Building the sample

The sample data service named `RDFSampleService` should be deployed using the instructions in Samples Setup.

Executing the sample

The sample service can be run using the TryIt tool, which is bundled with the WSO2 Data Services Server, or a code-generated java client sample as discussed in the Data Services Clients section.

Follow the steps below to demonstrate this functionality using the TryIt tool:

1. Log in to the management console of your server and click List under Services in the navigator. The `RDFSampleService` will be listed here.
2. Click Try this service to open the TryIt tool.
3. Select the "getAllMovieData" operation and click Send to execute the request. You will get details on all movies in the response as shown below.

4. Select the "getMoviesByGenre" operation to retrieve movie information based on the genre provided in the request.
Exposé data as RDF

A data service can expose data in RDF format by setting RDF as the output mapping type. This sample demonstrates how data is exposed in RDF format using a data service deployed in WSO2 DSS.

About the sample

This sample uses an excel sheet as the datasource. The *Products.xls* file, stored in the `<DSS_HOME>/samples/resources/` folder is used to get the data. This datasource contains details of the products manufactured in an automobile company. Products refer to cars/motorcycles. The data table has the following columns: "ID", "Name", "Classification" and "Price". The sample data service used in this demonstration (*ExposeAsRDFSample.dbs*) contains a REST resource which lists down all the products that are manufactured in the company.

See Data Services and Resources for details.

Building the sample

The sample data service named *ExposeAsRDFSample* should be deployed in the DSS instance using the instructions in Samples Setup.

Running the sample

The service can be invoked in REST-style via curl ([http://curl.haxx.se](http://curl.haxx.se)). The following command can be executed in the command line to retrieve the product info (stored in the `<DSS_HOME>/samples/resources/Products.xls` file) in RDF format.

```
curl -X GET
http://localhost:9763/services/samples/ExposeAsRDFSample/vehicles
```
OData Sample

Exposing a database as an OData service eliminates the need to define queries, operation etc. to access the information in the database. OData (Open Data Protocol) is an OASIS standard that defines the best practice for building and consuming RESTful APIs. You can easily expose databases as an OData service using WSO2 Data Services Server.

- About the sample
- Building the sample
- Executing the sample

About the sample

This OData sample was generated to work with the default h2 database (DATA_SERV_SAMP.h2.db), which used for samples in the DSS product. This database is stored in the <DSS_HOME>/samples/database directory. The details of this database are as follows:

Driver class: org.h2.Driver
JDBC URL: jdbc:h2:file:./samples/database/DATA_SERV_SAMP

Tables in the DB:
- CUSTOMERS
- ACCOUNTS
- PAYMENTS
You can find the DDL of these sql tables in `<DSS_HOME>/samples/sql/h2` folder.

Building the sample

The sample data service named `ODataSampleService` should be deployed and the server should be started using the instructions in `Samples Setup`.

Executing the sample

You can easily invoke the service using HTTP requests. See the following examples.

- To get the service document:

  ```
  GET https://localhost:9443/odata/ODataSampleService/default
  Accept : application/json
  ```

- To get the meta data of the service:

  ```
  GET https://localhost:9443/odata/ODataSampleService/default/$metadata
  Accept : application/xml
  ```

- To read customer details:

  ```
  GET https://localhost:9443/odata/ODataSampleService/default/CUSTOMERS
  Accept : application/json
  ```

- To add accounts details, send the HTTP request as shown below. In this example we will be creating an account ID named "1".

  ```
  POST https://localhost:9443/odata/ODataSampleService/default/ACCOUNTS
  Accept: application/json
  Content-Type: application/json
  Prefer : return=representation
  { "BALANCE" : 12.22 ,"ACCOUNTID" : 1 }
  ```

- To delete account details, send the HTTP request as shown below. In this example, we will be deleting the account ID "1".

```
• To update account details:

GET
https://localhost:9443/odata/ODataSampleService/default/ACCOUNTS(1)
Accept: application/json

• To invoke a particular query, send the HTTP request as shown below. In this example, we will receive the city and phone numbers of customers.

GET
https://localhost:9443/odata/ODataSampleService/default/CUSTOMERS?$select=CITY,PHONE
Accept: application/json

• To filter information based on a query, send the HTTP request as shown below. In this example, we will receive the employee details connected to the 'Nantes' city.

GET
https://localhost:9443/odata/ODataSampleService/default/CUSTOMERS?$filter=CITY eq 'Nantes'
Accept: application/json