CUSTOMER

Java Persistence API Models as OData Services
Document Version: - 2013-06-07

JPA Models as OData Services

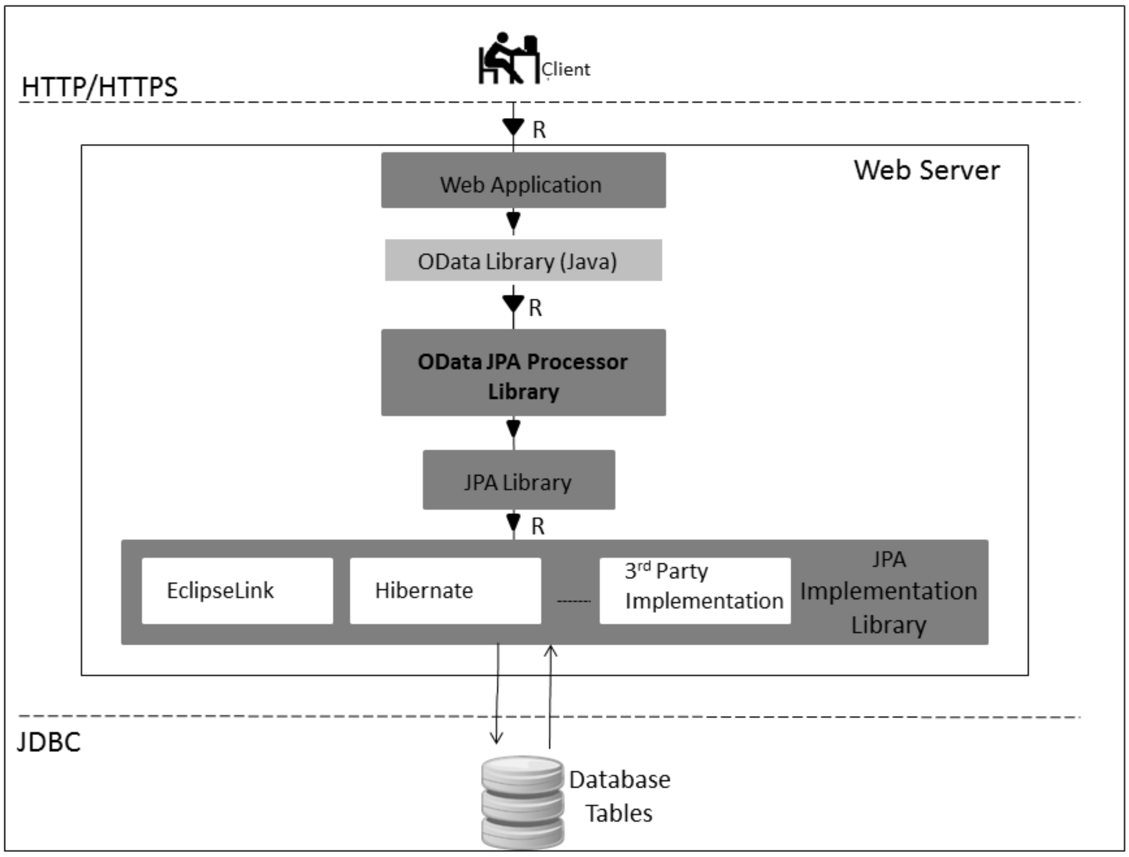
Table of Contents

1 Enabling JPA Models as OData Services

This section provides information on how to enable JPA models as OData Services and the related topics.

OData JPA Processor Library is a Java library for transforming Java Persistence API (JPA) models based on JPA specification into OData services. It is an extension of OData Library (Java) to enable Java developers to convert JPA models into OData services with minimal or no code.

OData JPA Processor Library has two main parts, Core and the API. API provides extension points using which you are allowed to customize / redefine the metadata and the runtime. The graphic below gives a high level overview of some of the important components involved in transforming JPA models into OData services.



Whenever a request is sent from the client, it passes through the OData Library (Java), where the request is processed and deserialized before it is sent to the OData JPA Processor Library. The OData Entities / Java Objects are processed and JPQL statements are sent to the JPA layer. JPA is defined in the `javax.persistence` package. You are free to use any of the JPA Providers (like Eclipse, Hibernate, OpenJPA and so on).

In this Section

- *JPA Glossary*

- *OData JPA Processor Library Supported Features*
- *Creating a New OData Project in Maven*
- *Redefining the Metadata*
- *Adding Function Imports as OData Services*

1.1 OData JPA Processor Library Glossary

Terminology for enabling JPA Models as OData Services Topic

The following provides an overview of the main terms and their definition as used in enabling JPA Models as OData services:

Table 1: Terms and Definitions

Terms	Definitions
complex type (synonym: ComplexType)	Structured types consisting of a list of properties but with no key. Can only exist as a property of a containing entity or as a temporary value.
entity data model (EDM)	Set of concepts that describe the structure of data regardless of its stored form.
function import (synonym: FunctionImport)	Describes a service operation in an entity data model.
metadata document	Complete XML representation of an Entity Data Model (1:1 relation).
navigation property (synonym: NavigationProperty)	Property of an entry that represents a link from this entry to one or more related entries. A navigation property is not a structural part of the entry to which it belongs.
Open Data Protocol (OData)	Standard Web protocol for querying and updating data. It applies and builds upon Web technologies such as HTTP, Atom Publising Protocol, and JSON to provide access to information from a variety of applications. For more information, see *OData*.
OData Library (Java)	It is a Java library which enables business applications to expose and consume data using OData protocol.
OData JPA Processor Library	It is a Java library (built on top of OData Library (Java)) for transforming Java persistence models based on JPA specifications into OData services with minimal code.
OData service	Develop OData services from JPA models using OData JPA Processor Library.
service document	Top-level XML representation of a consumption model (1:1 relation). It contains a list of entity sets. A

Terms	Definitions
	document that describes the location and capabilities of one or more entity sets.

Related Links

http://www.sapterm.com/

http://www.odata.org

1.2 OData JPA Processor Library Supported Features

OData JPA Processor Library supported features

This section contains the list of JPA features and the corresponding OData features that are supported when a Java Persisitence Model is converted to an OData service using OData JPA Processor Library.

JPA Feature	Description	OData Feature	Description	Status
Persistence Unit name	JPA Persistence Unit is a logical grouping of user defined persistable classes (entity classes, embeddable classes and mapped superclasses) with related settings.	Namespace	Namespace is a name that is defined on the schema and is subsequently used to prefix identifiers to form the namespace qualified name of a structural type. Naming Convention - The name of the namespace is same as the `Persistence Unit Name`.	Supported
Entity Annotated with @Entity	An entity class is a user defined Java class whose instances can be stored in the database.	EntityType	An EnityType has an unique identity, an independent existance and it forms the operational unit of consistency. Naming Convention - The name of an OData entity is same as the `JPA Model Entity Class` name.	Supported
Entity Key Annotated with: ● @Id ● @EmbeddedId ● @IdClass	Every entity object that is stored in the database has a primary key. Once assigned, the primary key cannot be modified. It represents the entity object as long as it exists in the database. A primary key can be simple as well as composite.	Key	An OData entity type must define an entity key or derive from a base type. Naming Convention - ● For JPA simple primary key, the key name is derived based on the name of the field which is annotated with @Id and the first character of the field name is taken in upper case. ● For JPA complex primary key @EmbeddedId, the field of the	Supported

JPA Feature	Description	OData Feature	Description	Status
	Simple primary key is annotated with @Id. A complex primary key can be annotatted with @EmbeddedId or @IdClass		complex type in JPA is expanded as simple EDM properties and references to those properties are created. The name of the key is derived as explained for simple primary key, above. ● For JPA complex primary key @IdClass, there is a set of fields defined in the entity class which are annotated with @Id. References to all the properties are created as key properties. The name of the key is derived as explained for simple primary key, above.	
Property Annotated with ● @Column ● @AttributeOverrides ● @Embedded ● @EmbeddedId	Defines the attributes of an entity. Properties are of three types: ● Single valued which includes simple java data types ● Multivalued which includes Collection types from package java.util (ArrayList, Vector, Stack, LinkedList, ArrayDeque, PriorityQueue, HashSet, LinkedHashSet, TreeSet , Map types from package java.util: HashMap, Hashtable, WeakHashMap, IdentityHashMap, LinkedHashMap, TreeMap and Properties and Arrays (including multi dimensional arrays)). ● Enum Types	Property	An Entity type can have one or more properties of scalar or complex type. Naming convention - The name of the EDM properties are derived based on the name of the field in the JPA Entity class. The first character of the field name is taken in upper case. ● All single valued JPA attributes are supported. ● Only multivalued properties representing an association are supported. ● Enums are not supported.	Partially Supported

JPA Feature	Description	OData Feature	Description	Status
Navigation Property Annotated with ● @ManyToOne ● @OneToMany ● @ManyToMany ● @OneToOne	Defines the association between two JPA entites.	Navigation Property	Navigation Property defines non-structural properties on an entity that allows navigation from one entity to another via a relationship. Navigation properties are created for each association object in the JPA entity class, provided the association is consistent. Naming convention - The name of the navigation property is derived based on the name of the target entity and the details (in the form, `<Name of the target enitiy>+Details`). The ToRole and FromRole of the navigation is the same as the role names of the association. The relationship name is the fully qualified name of the related association, that is , `<namespace>.<association_name>`.	Supported
EmbeddedTypes Annotated with ● @Embeddable ● @Embedded	Embeddable classes are user defined persistable classes that function as value types.	Complex Type	A complex type provides a mechanism to create declared properties with a rich structured payload. Naming convention - The name of the complex type is same as the `JPA Class name representing the complex type`.	Supported
Association Annotated with ● @ManyToOne ● @OneToMany ● @ManyToMany ● @OneToOne	Entity relationship define the relationship between two entities. Realtionships may be unidirectional or bidirectional.	Association	An association is a named independent relationship between two EntityType definitions. It can support different multiplicities at the two ends. Naming Convention - The name of the association is derived based on the names of one of the EDM entity type and the second entity type (in this form, `<name of one of the edm entity type>_<name of the second entity type>`). i Note	Partially Supported

JPA Feature	Description	OData Feature	Description	Status
			When a association has an additional annotation @JoinColumn and it does not have the Column or ReferencedColumnName parameter, then the association is not created.	
		Association End	For a given association, the End defines one side of the relationship. It defines what type is partcipating in the realtionship (multiplicity or the cardinality) and if there is any operation associations like Cascade delete. Naming convention - The association end's name is derived from the fully qualified name of the participating entity type, that is, `<namespace>.<entity type name>`	Support ed
		Association End Role	Naming convention - The name of the association end role is derived from the name of the corresponding entity type.	Support ed
Referential Constraint Annotated with • @JoinColumn (name = foreign property in the dependent side, referencedColu mnName = primary key in the principal entity) • @JoinColumns • @JoinTable	• @JoinColumn: Is used to specify a mapped column for joining an entity association • @JoinColumn: Defines mapping for the composite foreign keys. This annotation groups JoinColumn annotations for the same relationship. • @JoinTable: It is used in the mapping of associations. It is specified on the	Referential Constraint	Referential Constraint element can exist between the key of one entity type and the primitive property of another associated entity type. The two entity types are in a principal dependent relation. The referential constraint must specify which end is the principal role and which end is the dependent role. The principal role name and the dependent role name is the same as the role name in the association. The property references in the principal and dependent role are the same as OData property names. • If there is no @JoinColumn specified with an association, no referential constraint is created. This means the association is still	Partially Support ed

JPA Feature	Description	OData Feature	Description	Status
	owning side of an association .		created but without a referential constraint. ● For @JoinColumn without the name or referenced Column name parameter, no referential constraint is created. The association is also not created. ● @JoinColums- not supported ● @JoinTable- not supported	
		EntityContaine r	Entity Container is conceptually similar to a database or data source. It groups entityset, associationset and function import child elements that represent a data source. Naming convention - The name of the EntitiyContainer is derived based the names of the persistence unit name and the container (in this form `Persistence Unit Name +Container`).	Support ed
		EntitySet	Entity Set is a named set that can contain instances of a specified Entity type element. Naming convention - The name of the entity set is derived based on the names of the Association Entity Type and Set (in this form, `<Associated Entity Type+s>`). The entity type name is the fully qualified name of the entity type, that is, `<namespace>.<entity type name>`.	Support ed
		Association Set	An association set contains relationship instances of the specified association. The association specifies the entity type elements of the two end points, where as the association set specifies the entity set element that corresponds to these enity types. Naming convention - The name of an association set is derived based on the names of the association name and set (in this form `<Association`	Support ed

JPA Feature	Description	OData Feature	Description	Status
			`name>+Set`). The end roles are the same as that of the association and the end entity set names are same as the corresponding entity set name. The association name is the fully qualified nameof the association, that is, `<namespace>.<association name>`	
Custom Operation	Java Methods annotated with @FunctionImport, @Parameter can be transformed into OData Function Imports.	Function Import	Naming Convention - By default the Java method name is taken as the Function Import name. However, this can be overriden by the Java annotation attribute "name" of @FunctionImport. For example: @FunctionImport(name="Process") int process( ){ }	Support ed
@Transient	Fields in a JPA Entity can be annotated with @Transient. Such fields in the JPA model is not to be persisted in the database. They are used for internal logic and processing.			Not Support ed
Entity Inheritence	JPA Entity B extends JPA Entity A.	Base Type		Not Support ed

Runtime

The list of runtime features that are supported / not supported is given in the table.

OData Feature	Status
$orderby	Supported
$top	Supported
$skip	Supported
$filter	Supported
$inlinecount	Supported
$format	Supported
$select	Supported
$expand	Supported

OData Feature	Status
$count	Supported
$value	Not Supported
$links	Not Supported
Custom Query Options	Not Supported

1.3 Creating Web Application Project for Transforming JPA Models into OData Services in Maven

Information on creating a web applicaion project for transforming JPA models into OData services in Maven using OData JPA Processor Library.

Procedure on how to create a web applicationin in Maven for transofrming JPA Models into OData Services using OData JPA Processor Library is provided in this section.

The table gives the list of Maven dependencies you need to include in the POM.xml of your application.

Note

The following dependencies are applicable for an application using ECLIPSELINK as the JPA Provider and HSQLDB as the database. However, you are free to use any JPA provider (like Hibernate,OpenJPA and so on) and database of your choice.

Group ID	Artifact ID	Version (Tested)
com.sap.core.odata	com.sap.core.odata.api	0.4.0
com.sap.core.odata	com.sap.core.odata.core	0.4.0
javax.servlet	servlet-api	2.5
org.apapche.cxf	cxf-rt-frontend-jaxrs	2.7.0
org.slf4j	slf4j-api	1.7.1
org.eclipse.persistence	eclipselink	2.1.2
org.eclipse.persistence	javax.persistence	2.0.3
org.hsqldb	hsqldb	1.8.0.10
com.sap.core.odata	com.sap.core.odata.processor.core	0.3.0
com.sap.core.odata	com.sap.core.odata.processor.api	0.3.0
com.sap.core.odata	com.sap.core.odata.api.annotation	0.3.0

1. Create a Dynamic Web Application project from scratch:
   a) In the command prompt, enter the maven command given here (change the DgroupId and DartifactId as per your requirement)

```
mvn archetype:generate -DgroupId=foo -DartifactId=salesorderprocessing.app -
DarchetypeArtifactId=maven-archetype-webapp
```

Maven generates the file system structure for a web application project including a basic POM.xml. This step is completed by creating a Java source folder.

b) Create a folder by name 'Java' in the path 'src/man/'.

2. Tailor POM.xml - POM.xml should be modified for adding dependencies like OData Library (Java) and OData JPA Processor Library and configuring the Eclipse plugin for the generation of project files.

   Open POM.xml and replace the existing content with the following:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
 <modelVersion>x.x.x</modelVersion>
 <artifactId>salesorderprocessing.app</artifactId>
 <packaging>war</packaging>
 <url>http://maven.apache.org</url>
 <build>
  <finalName>salesorderprocessing.app</finalName>
   <plugins>
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-eclipse-plugin</artifactId>
    <version>x.x</version>
    <configuration>
     <wtpversion>x.x</wtpversion>
     <projectNameTemplate>[artifactId]-[version]</projectNameTemplate>
    </configuration>
   </plugin>
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>x.x.x</version>
    <configuration>
     <source>1.6</source>
     <target>1.6</target>
    </configuration>
   </plugin>
  </plugins>
 </build>
 <dependencies>
  <!-- OData Library (Java) Support -->
  <dependency>
   <groupId>com.sap.core.odata</groupId>
   <artifactId>com.sap.core.odata.api</artifactId>
   <version>x.x.x</version>
  </dependency>
  <dependency>
   <groupId>com.sap.core.odata</groupId>
   <artifactId>com.sap.core.odata.core</artifactId>
   <version>x.x.x</version>
  </dependency>
  <!-- required because of auto detection of web facet 2.5 -->
  <dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>servlet-api</artifactId>
   <version>x.x</version>
   <scope>provided</scope>
  </dependency>
  <dependency>
   <groupId>org.apache.cxf</groupId>
   <artifactId>cxf-rt-frontend-jaxrs</artifactId>
   <version>x.x.x</version>
  </dependency>
```

```
   <!-- JPA Support -->
   <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>eclipselink</artifactId>
    <version>x.x.xversion>
   </dependency>
   <dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>javax.persistence</artifactId>
    <version>x.x.xversion>
   </dependency>
   <dependency>
    <groupId>org.hsqldb</groupId>
    <artifactId>hsqldb</artifactId>
    <version>x.x.x.xversion>
   </dependency>
   <dependency>
    <groupId>com.sap.core.odata</groupId>
    <artifactId>com.sap.core.odata.processor.core</artifactId>
    <version>x.x.x</version>
   </dependency>
   <dependency>
    <groupId>com.sap.core.odata</groupId>
    <artifactId>com.sap.core.odata.processor.api</artifactId>
    <version>x.x.x</version>
   </dependency>
  </dependencies>
  <name>OData Processors - JPA Reference Scenario</name>
  </project>
```

3. Generate Eclipse Project - This generates the files to open the maven project inside Eclipse. Execute the following maven command in the command prompt:

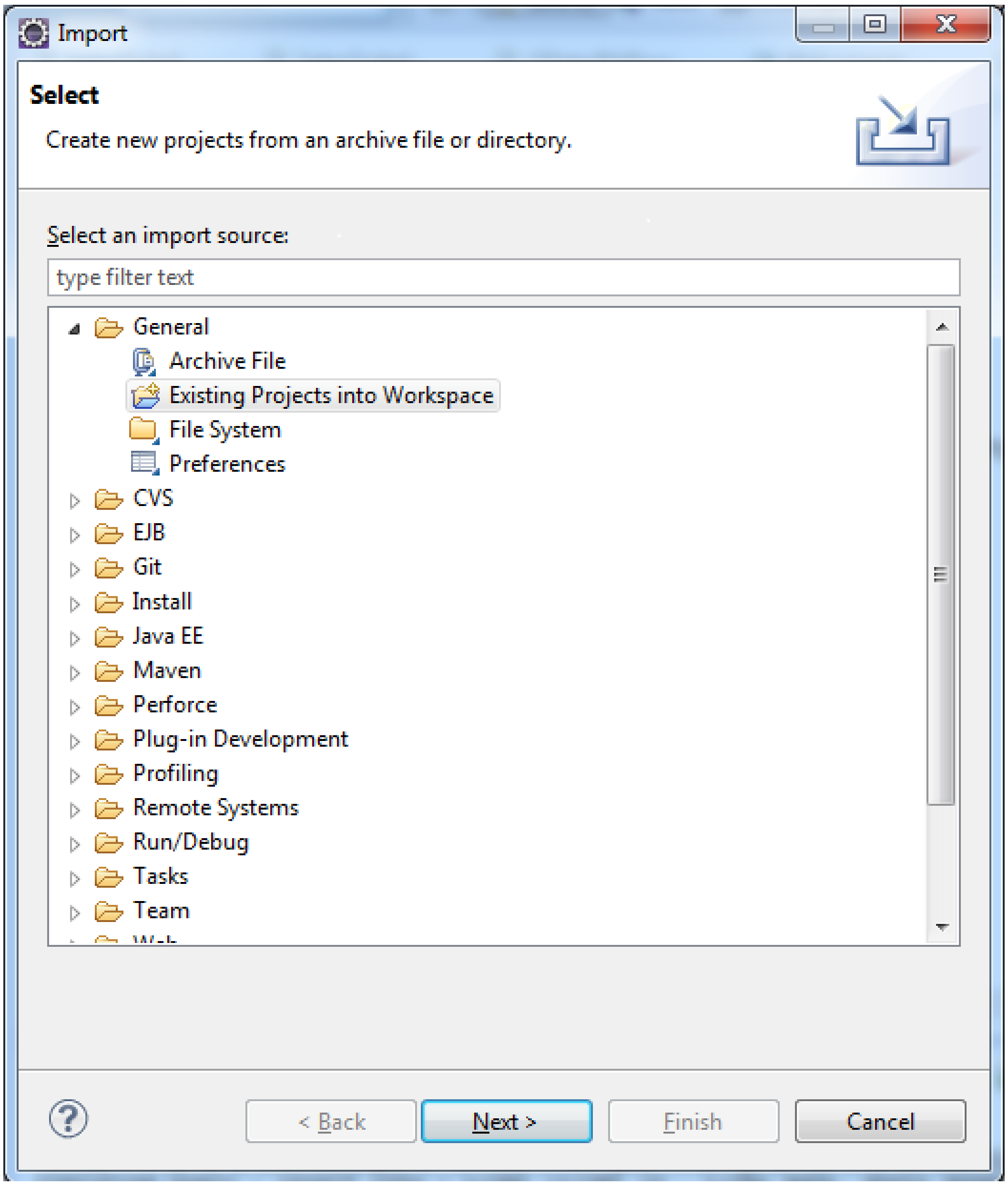
   Code Syntax

   ```
   mvn eclipse:clean eclipse:eclipse
   ```

   As a result, Maven generates a `.project` and a `.classpath` file which can be imported to Eclipse. It should appear as a Dynamic Web Application project which is ready to use and deployable to any web application server.

4. Import Project into Eclipse:

   a) Open the project created in Eclipse: Go to *File* > *Import* > *General* > *Exsiting Project into Workspace*.



b) Click *Next*. In the *Select Root Directory* field, click browse and select the maven project folder you created above, click *OK*.

c) Select the folder from the project list and click *Finish*.

5. JPA Implementation - In the web project, create persistence entities in the folder `src/main/java` by annotating them with JPA annotations. Configure persistence entities using `persistence.xml`. The configuration file should be placed under the folder `src/main/resources/META-INF/`.

6. Implement an OData Service - The project is now ready to expose OData services. Service Factory provides a means for initializing Entity Data Model (EDM) Provider and OData JPA Processors with OData JPA Contexts. Following are the steps for implementing a Service Factory:

a) Choose the project folder, go to context menu ▶ *New* ▶ *Class* ▶.

b) In the *Java Class* window, enter the package and a name for the Java class.

c) Browse for ODataJPAServiceFactory in the *Super Class* field.

d) Declare persistence unit name as class variable. For example, `private static final String PUNIT_NAME = "persistenceUnitName";`

e) Implement the abstract method `initializeJPAContext` in the factory class. Here is the code snippet:

```
ODataJPAContext oDataJPAContext= this.getODataJPAContext();
EntityManagerFactory emf = Persistence.createEntityManagerFactory(PUNIT_NAME);
oDataJPAContext.setEntityManagerFactory(emf);
oDataJPAContext.setPersistenceUnitName(PUNIT_NAME);
```

7. Configure the web application:

a) Configure the web application as shown below by adding the following servlet configuration to `web.xml`. The Service factory which was implemented is configured in the `web.xml` of the ODataApplication as one of the init parameters.

b) Replace **`<'com.sap.core.odata.processor.ref.JPAReferenceServiceFactory'>`** in the following XML with the class name you created in the previous step:

```
<servlet>
  <servlet-name>JPARefScenarioServlet</servlet-name>
  <servlet-class>org.apache.cxf.jaxrs.servlet.CXFNonSpringJaxrsServlet</
servlet-class>
  <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>com.sap.core.odata.core.rest.app.ODataApplication</param-
value>
  </init-param>
  <init-param>
      <param-name>com.sap.core.odata.service.factory</param-name>
      <param-value>foo.SalesOrderProcessingFactory</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>JPARefScenarioServlet</servlet-name>
  <url-pattern>/<ServiceName>.svc/*</url-pattern>
</servlet-mapping>
```

In the <url-pattern> tag, change the <ServiceName> with user specific service name in the pattern provided.

8. Save the settings.

1.4 Redefining Metadata

Steps to redefine the metadata of the OData service created from the JPA models.

The OData services created from JPA models using OData JPA Processor Library derives the names for its elements from Java Persistence Entity elements. These derived (default) names can be redefined using JPA EDM mapping models. JPA EDM Mapping model can be used to redefine:

1. Schema Namespace Name
2. Entity Type Names
3. Entity Set Names
4. Property Names
5. Navigation Property Names
6. Complex Type Names

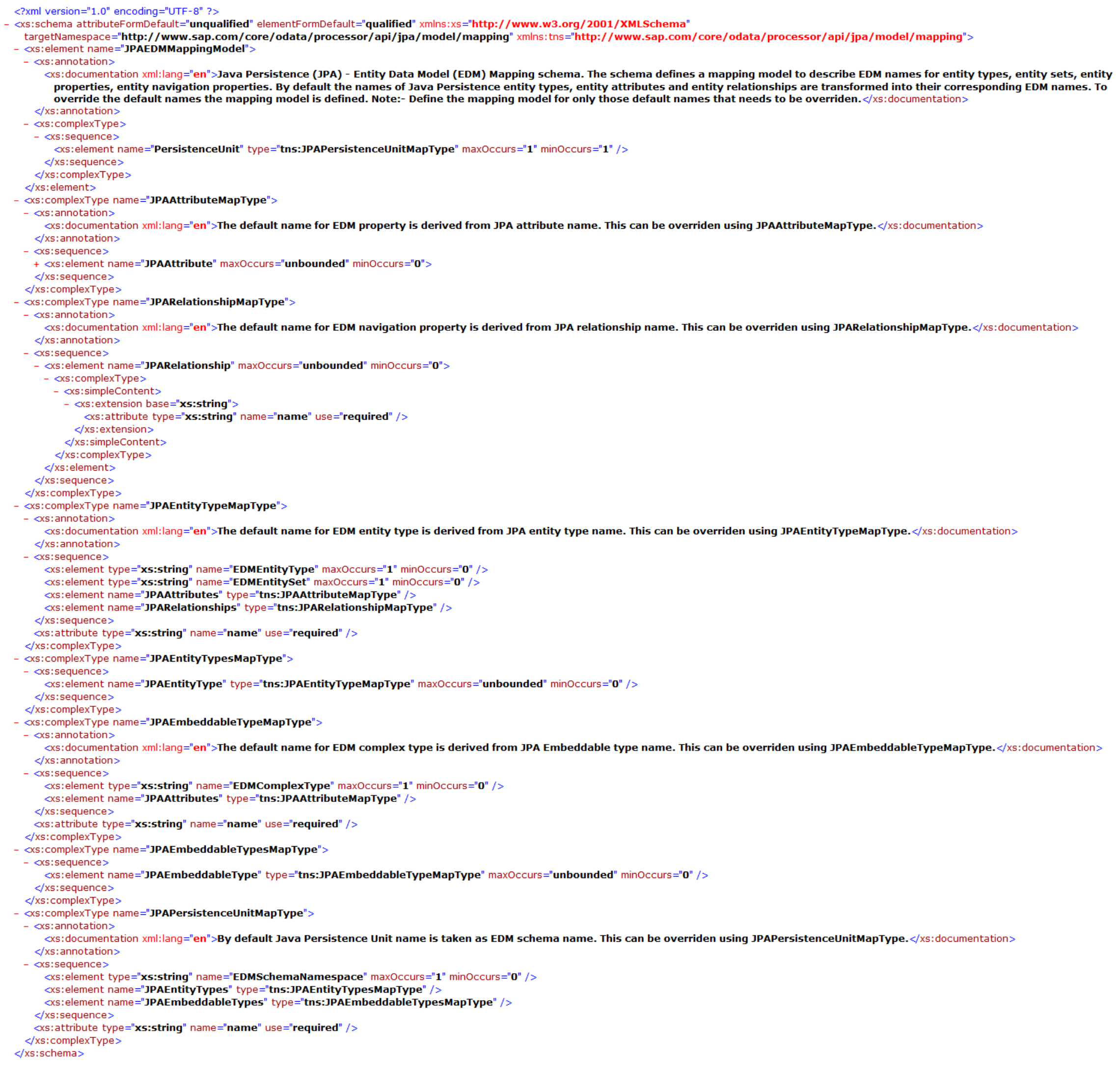
The OData JPA Processor Library applies certain naming rules to derive the names for the above OData elements by default. Here are the rules:

1. Schema Namespace Name is derived from Java Persistence Unit Name.
2. Entity Type Names are derived from Java Persistence Entity Type Names.
3. Entity Set Names are derived from EDM Entity Type Names suffixed with character "s".
4. Property Names are derived from Java Peristence Entity Attribute Names. The initial character in the property name is converted to an uppercase character.
5. Navigation Property Names are derived from Java Perisistence attribute name representing relationships. The navigation property name is suffixed with the word "Details".
6. Complex Type Names are derived from Java Persistence Embeddable type names.

Note

The names generated by applying the above rules can be overriden using JPA EDM Mapping models. JPA EDM mapping model can be maintaned as an XML document according to the schema.

1. Create a JPA EDM Mapping model XML according to the schema given in the screenshot below. In the XML, maintain the mapping only for those elements that needs to be redefined. For example, if JPA Entity Type A's name has to be redefined, then maintain an EDM name for the same.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.sap.com/core/odata/processor/api/jpa/model/mapping" xmlns:tns="http://www.sap.com/core/odata/processor/api/jpa/model/mapping">
  - <xs:element name="JPAEDMMappingModel">
    - <xs:annotation>
        <xs:documentation xml:lang="en">Java Persistence (JPA) - Entity Data Model (EDM) Mapping schema. The schema defines a mapping model to describe EDM names for entity types, entity sets, entity
          properties, entity navigation properties. By default the names of Java Persistence entity types, entity attributes and entity relationships are transformed into their corresponding EDM names. To
          override the default names the mapping model is defined. Note:- Define the mapping model for only those default names that needs to be overriden.</xs:documentation>
      </xs:annotation>
    - <xs:complexType>
      - <xs:sequence>
          <xs:element name="PersistenceUnit" type="tns:JPAPersistenceUnitMapType" maxOccurs="1" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:complexType name="JPAAttributeMapType">
    - <xs:annotation>
        <xs:documentation xml:lang="en">The default name for EDM property is derived from JPA attribute name. This can be overriden using JPAAttributeMapType.</xs:documentation>
      </xs:annotation>
    - <xs:sequence>
      + <xs:element name="JPAAttribute" maxOccurs="unbounded" minOccurs="0">
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="JPARelationshipMapType">
    - <xs:annotation>
        <xs:documentation xml:lang="en">The default name for EDM navigation property is derived from JPA relationship name. This can be overriden using JPARelationshipMapType.</xs:documentation>
      </xs:annotation>
    - <xs:sequence>
      - <xs:element name="JPARelationship" maxOccurs="unbounded" minOccurs="0">
        - <xs:complexType>
          - <xs:simpleContent>
            - <xs:extension base="xs:string">
                <xs:attribute type="xs:string" name="name" use="required" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="JPAEntityTypeMapType">
    - <xs:annotation>
        <xs:documentation xml:lang="en">The default name for EDM entity type is derived from JPA entity type name. This can be overriden using JPAEntityTypeMapType.</xs:documentation>
      </xs:annotation>
    - <xs:sequence>
        <xs:element type="xs:string" name="EDMEntityType" maxOccurs="1" minOccurs="0" />
        <xs:element type="xs:string" name="EDMEntitySet" maxOccurs="1" minOccurs="0" />
        <xs:element name="JPAAttributes" type="tns:JPAAttributeMapType" />
        <xs:element name="JPARelationships" type="tns:JPARelationshipMapType" />
      </xs:sequence>
      <xs:attribute type="xs:string" name="name" use="required" />
    </xs:complexType>
  - <xs:complexType name="JPAEntityTypesMapType">
    - <xs:sequence>
        <xs:element name="JPAEntityType" type="tns:JPAEntityTypeMapType" maxOccurs="unbounded" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="JPAEmbeddableTypeMapType">
    - <xs:annotation>
        <xs:documentation xml:lang="en">The default name for EDM complex type is derived from JPA Embeddable type name. This can be overriden using JPAEmbeddableTypeMapType.</xs:documentation>
      </xs:annotation>
    - <xs:sequence>
        <xs:element type="xs:string" name="EDMComplexType" maxOccurs="1" minOccurs="0" />
        <xs:element name="JPAAttributes" type="tns:JPAAttributeMapType" />
      </xs:sequence>
      <xs:attribute type="xs:string" name="name" use="required" />
    </xs:complexType>
  - <xs:complexType name="JPAEmbeddableTypesMapType">
    - <xs:sequence>
        <xs:element name="JPAEmbeddableType" type="tns:JPAEmbeddableTypeMapType" maxOccurs="unbounded" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="JPAPersistenceUnitMapType">
    - <xs:annotation>
        <xs:documentation xml:lang="en">By default Java Persistence Unit name is taken as EDM schema name. This can be overriden using JPAPersistenceUnitMapType.</xs:documentation>
      </xs:annotation>
    - <xs:sequence>
        <xs:element type="xs:string" name="EDMSchemaNamespace" maxOccurs="1" minOccurs="0" />
        <xs:element name="JPAEntityTypes" type="tns:JPAEntityTypesMapType" />
        <xs:element name="JPAEmbeddableTypes" type="tns:JPAEmbeddableTypesMapType" />
      </xs:sequence>
      <xs:attribute type="xs:string" name="name" use="required" />
    </xs:complexType>
  </xs:schema>
```

2. Store the JPA EDM Mapping model XML in `webapp` folder of Java web application. Make sure the file is placed under the applications root folder (relative to class path ../../WEB-INF/classes) when the application is deployed on web server.
3. Pass the XML name into ODataJPAContext. In the method `initializeODataJPAContext`, pass the name of the XML document as shown below:

   ```
   oDataJPAContext.setJPAEdmNameMappingModel(<name of xml file>);
   ```

4. Compile, deploy and run the web application in a web server.

Example

Sample JPA EDM Mapping Model

```
<?xml version="1.0" encoding="UTF-8"?>
<JPAEDMMappingModel
 xmlns="http://www.sap.com/core/odata/processor/api/jpa/model/mapping" >
 <PersistenceUnit name="salesorderprocessing">
  <EDMSchemaNamespace>SalesOrderProcessing</EDMSchemaNamespace>
  <JPAEntityTypes>
   <JPAEntityType name="SalesOrderHeader">
```

```
    <EDMEntityType>SalesOrder</EDMEntityType>
    <EDMEntitySet>SalesOrders</EDMEntitySet>
    <JPAAttributes>
     <JPAAttribute name="soId">ID</JPAAttribute>
     <JPAAttribute name="netAmount">NetAmount</JPAAttribute>
     <JPAAttribute name="buyerAddress">BuyerAddressInfo</JPAAttribute>
    </JPAAttributes>
    <JPARelationships>
     <JPARelationship name="salesOrderItem">SalesOrderLineItemDetails</
JPARelationship>
     <JPARelationship name="notes">NotesDetails</JPARelationship>
    </JPARelationships>
   </JPAEntityType>
   <JPAEntityType name="SalesOrderItem">
    <EDMEntityType>SalesOrderLineItem</EDMEntityType>
    <EDMEntitySet>SalesOrderLineItems</EDMEntitySet>
    <JPAAttributes>
     <JPAAttribute name="liId">ID</JPAAttribute>
     <JPAAttribute name="soId">SalesOrderID</JPAAttribute>
    </JPAAttributes>
    <JPARelationships>
     <JPARelationship name="salesOrderHeader">SalesOrderHeaderDetails</
JPARelationship>
     <JPARelationship name="materials">MaterialDetails</JPARelationship>
    </JPARelationships>
   </JPAEntityType>
  </JPAEntityTypes>
  <JPAEmbeddableTypes>
   <JPAEmbeddableType name="Address">
    <EDMComplexType>AddressInfo</EDMComplexType>
     <JPAAttributes>
      <JPAAttribute name="houseNumber">Number</JPAAttribute>
      <JPAAttribute name="streetName">Street</JPAAttribute>
     </JPAAttributes>
   </JPAEmbeddableType>
  </JPAEmbeddableTypes>
 </PersistenceUnit>
</JPAEDMMappingModel>
```

1.5 Adding Function Imports to OData Services

Information on how to enable custom operations as function imports.

This section explains how to enable custom operations as function imports. Function imports are used to perform custom operations on a JPA entity in addition to CRUD operations. For example, consider a scenario where you would like to check the availability of an item to promise on the sales order line items. ATP check is a custom operation that can be exposed as a function import in the schema of OData service.

1. Create a dependency to EDM Annotation Project. This is required to use the annotations that are defined in the project.

```
<dependency>
    <groupId>com.sap.core.odata</groupId>
    <artifactId>com.sap.core.odata.api.annotation</artifactId>
    <version>x.x.x</version>
    <scope>provided</scope>
</dependency>
```

2. Create a Java class and annotate the Java methods implementing custom operations with Function Import and Parameter Java annotations as shown below. Java methods can be created in JPA entity types and these methods can be annotated with EDM annotations for function import.

```
package com.sap.core.odata.processor.ref;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.Persistence;
import javax.persistence.Query;

import com.sap.core.odata.api.annotation.edm.Facets;
import com.sap.core.odata.api.annotation.edm.FunctionImport;
import com.sap.core.odata.api.annotation.edm.FunctionImport.Multiplicity;
import com.sap.core.odata.api.annotation.edm.FunctionImport.ReturnType;
import com.sap.core.odata.api.annotation.edm.Parameter;
import com.sap.core.odata.api.annotation.edm.Parameter.Mode;
import com.sap.core.odata.api.annotation.edmx.HttpMethod;
import com.sap.core.odata.api.annotation.edmx.HttpMethod.Name;
import com.sap.core.odata.processor.ref.jpa.Address;
import com.sap.core.odata.processor.ref.jpa.SalesOrderHeader;
import com.sap.core.odata.processor.ref.jpa.SalesOrderItem;

public class SalesOrderHeaderProcessor {

    private EntityManager em;

    public SalesOrderHeaderProcessor() {
        em = Persistence.createEntityManagerFactory("salesorderprocessing")
                .createEntityManager();
    }

    @SuppressWarnings("unchecked")
    @FunctionImport(name = "FindAllSalesOrders", entitySet = "SalesOrders",
returnType = ReturnType.ENTITY_TYPE, multiplicity = Multiplicity.MANY)
    public List<SalesOrderHeader> findAllSalesOrders(
            @Parameter(name = "DeliveryStatusCode", facets = @Facets(maxLength =
2)) String status) {

        Query q = em
                .createQuery("SELECT E1 from SalesOrderHeader E1 WHERE
E1.deliveryStatus = '"
                        + status + "'");
        List<SalesOrderHeader> soList = (List<SalesOrderHeader>) q
                .getResultList();
        return soList;
    }

    @FunctionImport(name = "CheckATP", returnType = ReturnType.SCALAR,
multiplicity = Multiplicity.ONE, httpMethod = @HttpMethod(name = Name.GET))
    public boolean checkATP(
            @Parameter(name = "SoID", facets = @Facets(nullable = false), mode =
Mode.IN) Long soID,
            @Parameter(name = "LiId", facets = @Facets(nullable = false), mode =
Mode.IN) Long lineItemID) {
        if (soID == 2L)
            return false;
        else
            return true;
    }

    @FunctionImport(returnType = ReturnType.ENTITY_TYPE, entitySet =
"SalesOrders")
    public SalesOrderHeader calculateNetAmount(
            @Parameter(name = "SoID", facets = @Facets(nullable = false)) Long
soID) {
        Query q = em
                .createQuery("SELECT E1 from SalesOrderHeader E1 WHERE E1.soId =
"
                        + soID + "l");
        if (q.getResultList().isEmpty())
            return null;
        SalesOrderHeader so = (SalesOrderHeader) q.getResultList().get(0);
        double amount = 0;
        for (SalesOrderItem soi : so.getSalesOrderItem()) {
            amount = amount
                    + (soi.getAmount() * soi.getDiscount() * soi.getQuantity());
        }
        so.setNetAmount(amount);
        return so;
    }

    @SuppressWarnings("unchecked")
    @FunctionImport(returnType = ReturnType.COMPLEX_TYPE)
    public Address getAddress(
            @Parameter(name = "SoID", facets = @Facets(nullable = false)) Long
soID) {
        Query q = em
                .createQuery("SELECT E1 from SalesOrderHeader E1 WHERE E1.soId =
"
                        + soID + "l");
        List<SalesOrderHeader> soList = (List<SalesOrderHeader>) q
                .getResultList();
        if (!soList.isEmpty())
            return soList.get(0).getBuyerAddress();
        else
            return null;
    }

    /*
     * This method will not be transformed into Function Import Function Import
     * with return type as void is not supported yet.
     */
    @FunctionImport(returnType = ReturnType.NONE)
    public void process(
            @Parameter(name = "SoID", facets = @Facets(nullable = false)) Long
soID) {
        return;
    }

}
```

3. Create a Java class by implementing the interface `com.sap.core.odata.processor.api.jpa.model.JPAEdmExtension` to register the annotated Java methods.

```
public class SalesOrderProcessingExtension implements JPAEdmExtension {

    public void extend(JPAEdmSchemaView view){
        view.registerOperations(SalesOrderHeaderProcessor.class,null);
    }
}
```

Note

Use the method `extend` to register the list of classes and the methods within the class that needs to be exposed as Function Imports. If the second parameter is passed null, then the OData JPA Processor Library would consider all the annotated methods within the class for Function Import. However, you could also restrict the list of methods that needs to be transformed into function imports within a Java class by passing an array of Java method names as the second parameter.

4. Register the class created in step 3 with `ODataJPAContext` as shown below. The registration can be done during the initialization of `ODataJPAContext` in OData JPA Service Factory along with initializing persistence unit name, entity manager factory instance and optional mapping model.

   ```
   oDataJPAContext.setJPAEdmExtension((JPAEdmExtension) new
   SalesOrderProcessingExtension());
   ```

   Note

   You must register the class because the OData JPA Processor Library should be informed about the list of Java methods that it needs to process in a project. If we do not register, then OData JPA Processor Llibrary have to scan all the classes and the methods in the Java project looking for EDM annotations. In order to avoid such overload, it is mandatory to specificy the list of Java methods that shall be transformed into function imports in a class.

www.sap.com/contactsap