RFC-232: JPA Service Enhancements

Draft

13 Pages

Abstract

The OSGi JPA Service provides a modular mechanism for using JPA within an OSGi framework. Version 1.0 of the JPA Service was introduced in the OSGi Enterprise Specification Release 4.2. Since this time there have been new versions of the JPA Specification, and new related OSGi specifications such as the Transaction Control Service. This RFP gathers updated requirements to ensure the continued usefulness of the JPA service.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 10.1.

```
Source code is shown in this typeface.
```

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	October 18 2016	Initial version of the RFC
0.1	November 29 2016	Updated to remove the unnecessary part of the API contract

1 Introduction

The Java Persistence API may be used in Java EE, Java SE and OSGi as a standard API for O-R mapping. Using JPA in OSGi requires the use of the OSGi JPA Service. This service discovers persistence units contained inside persistence bundles and registers EntityManagerFactory and/or EntityManagerFactoryBuilder services for them.

This document describes the updated requirements that must be met in order for the OSGi JPA service to be used with newer versions of the JPA Service specification. It also details a number of necessary enhancements required in order for the JPA service to be easily integrated with other OSGi specifications, such as the Transaction Control Service. For example:

- Providing fully configured datasources to the EntityManagerFactoryBuilder
- Changing the Transaction type of a persistence unit via the EntityManagerFactoryBuilder

- Requiring that an EntityManagerFactoryBuilder is registered even in the case of a complete persistence unit

2 Application Domain

Persistence is a critical layer in many Java applications. It is used to read and write to the relational database through JDBC drivers without having to embed database-specific code in the application..

Virtually any application can use JPA to meet its ORM persistence needs, thus the application domain is almost as wide and deep as the application space is itself. Certainly the traditional enterprise applications like web, business and IT applications commonly use JPA and similar persistence solutions to connect to RDB technology.

JPA runs in two different modes, depending upon the environment that it is being used in. When running in Java SE, or a non-container environment, the bootstrap Persistence API class is used with local transactions. When used in a Java EE container, or any other compliant JPA container host, the container provides persistence provider detection and loading, as well as injection services of certain useful JPA resources.

JPA in an OSGi framework supports both of these two modes using OSGi services, the Persistence bootstrap class can discover these OSGi services, as can injection containers such as Declarative Services and OSGi Blueprint.

2.1 Terminology + Abbreviations

- JPA: Java Persistence API
- Persistence Unit: The JPA unit(s) of configuration stored in an external persistence XML file
- Resource Local Persistence Unit: A Persistence Unit that is configured to use local transactions
- JTA Persistence Unit: A Persistence Unit that has a life cycle which is tied to an ongoing transaction
- Entity Manager: The main entry point object to provide JPA operations
- Java EE: Java Enterprise Edition, the enterprise Java development and runtime platform
- Java SE: Java Standard Edition, the standard Java development and runtime platform
- JPA Container: Any container that supports and implements the container SPI, invokes JPA providers, and injects EntityManager and EntityManagerFactory instances into client classes
- JPA Provider: A vendor implementation of the JPA interfaces
- Managed Class: Class that can be mapped to the database, viz. an entity, embeddable or mapped superclass
- SPI: Service Provider Interface, or an interface that can be used to invoke a JPA Provider offering a specific service
- Persistence Provider: see JPA Provider

3 Problem Description

The OSGi JPA Service is a flexible way to use JPA in an OSGi environment, and solves many problems. Persistence Bundles are processed by a JPA Service extender, and any persistence units that they define gain associated EntityManagerFactoryBuilder services.

Many use cases are adequately addressed by the OSGi JPA Service 1.0, however there are a number of problems that have emerged over the years since its release.

3.1 Problems

3.1.1 JPA Base version

The OSGi JPA Service version 1.0 defines support for JPA version 1.0, however the current version of the JPA specification is 2.1. Modern applications make extensive use of these new JPA features, and the OSGi JPA service should also support them.

It should also be noted that despite the increase in the major version of the JPA specification there has been no breaking change to the API. Semantic versioning for the API is therefore difficult, and it can be challenging to ensure that JPA clients and providers share a common view of the API.

3.1.2 Requiring the JPA Extender

The OSGi JPA Service version 1.0 defines an extender model using the Meta-Persistence header, however there is no associated requirement/capability for this extender. This means that automatic resolution and deployment of systems using the JPA service is difficult.

3.1.3 Incomplete/Complete Persistence Units

The OSGi JPA service does not require that a persistence unit be complete (i.e. contain the full configuration for the persistence unit (see JPA service spec 127.4.7)). Additional configuration and/or configuration overrides can be provided using the EntityManagerFactoryBuilder service. An incomplete persistence unit is one that does not provide database connection information.

The lifecycle of incomplete Persistence units is poorly defined in the current specification. For example if an incomplete persistence unit is detected it is unclear whether an EntityManagerFactory should be registered. As this service is unusable without the database connection it should clearly not be available.

In general incomplete persistence units are preferable to complete persistence units as they allow the database configuration to be completely defined by configuration, rather than being packaged in to the persistence bundle. The Specification chapter should more clearly describe this model, along with any recommendations.

3.1.4 Standard properties

The JPA 2.0 and 2.1 specifications have defined a number of standard property names for things such as the database metadata, and other persistence unit configuration. The JPA Service specification should be clear that

such properties can be passed to an EntityManagerFactoryBuilder and will take precedence over any properties defined in the persistence unit. This can, for example, be used to supply a DataSource to an incomplete (or complete) persistence unit, or to change the transaction type of the Persistence unit from its default type.

4 Requirements

JPA 0010: The solution MUST allow persistence bundles and clients to make use of JPA version 2.1 specification features

JPA 0020: The solution MUST enable persistence bundles to continue working with future backward compatible versions of the JPA specification

JPA 0030: The solution MUST enable OSGi JPA applications to be assembled using a resolver operation in the same way that other OSGi applications can be assembled

JPA 0040: The solution MUST allow for database configuration information to be provided at runtime

JPA 0050: The solution MUST allow for a configured DataSource to be provided to the EntityManagerFactoryBuilder

JPA 0060: The solution MUST allow the transaction type of the persistence unit to be changed when using the EntityManagerFactoryBuilder

5 Technical Solution

The OSGi JPA service is built around colaboration between three different actors. The persistence bundle is written by the user and contains a persistence descriptor and one or more managed types. The JPA persistence provider is an implementation of the JPA specification, and uses the persistence descriptor and managed type(s) to provide Object Relational Mapping against a database. The OSGi JPA Service is responsible for binding a persistence bundle to a persistence provider, and for ensuring that the JPA resources are registered as OSGi Services.

5.1 Updating the JPA API Version

Version 1.0 of the JPA service makes use of JPA 1.0. There have been two subsequent releases of the JPA specification which have both greatly increased the functions available to JPA clients. This RFC therefore proposes increasing the base level of JPA supported by the JPA service to the current JPA specification version, which is JPA 2.1.

Unfortunately, as is the case with many JSRs, the version changes associated with the JPA specification updates have not adhered to semantic versioning rules and so consuming the API can be error prone.

5.1.1 The JPA API contract

In order to permit JPA clients to rely on features from newer versions of JPA there needs to be a defined contract upon which the client can rely, otherwise a JPA 1.0 compatible client cannot declare a dependency which also accepts the backward compatible JPA 2.0 API. For JPA the following three contracts exist (See [3]. for contract definitions)

- osgi.contract;osgi.contract=JavaJPA;version:Version=1;uses:=”javax.persistence,javax.persistence.spi”
- osgi.contract;osgi.contract=JavaJPA;version:Version=2;uses:=”javax.persistence, javax.persistence.criteria, javax.persistence.metamodel, javax.persistence.spi”
- osgi.contract;osgi.contract=JavaJPA;version:Version=2.1;uses:=”javax.persistence, javax.persistence.criteria, javax.persistence.metamodel, javax.persistence.spi”

These contracts provide an excellent basis for clients, as API providers can advertise the version(s) of the API that they support, including backward compatible versions.

5.1.2 Standard JPA property names

The JPA 2.1 specification adds a significant number of standard property names. These properties are used both for runtime control, and also for configuring JPA persistence units as they are created.

The EntityManagerFactoryBuilder service must support the defined property names as per the JPA specification. In most cases this will be accomplished by passing the values directly to the Persistence Provider.

The following values are identified in the JPA 2.1 Specification [4].

Property name(s)	**Purpose**
javax.persistence.jdbc.driver javax.persistence.jdbc.url javax.persistence.jdbc.user javax.persistence.jdbc.password javax.persistence.dataSource	Properties used for binding a Database driver. If the JDBC driver class name is supplied as a String then the JPA Service implementation must attempt to load the driver class using the persistence bundle’s classloader. If this fails the JPA Service must look for a suitable JDBC Service implementation to create the Driver. If this also fails then an exception must be thrown. If bound to a JDBC service implementation the EntityManagerFactoryBuilder must be destroyed and re-registered if the JDBC service is unregistered.
javax.persistence.schema-generation.create-script-source javax.persistence.schema-generation.drop-script-source javax.persistence.sql-load-script-source	Controlling initial database creation and population

javax.persistence.schema-generation.database.action javax.persistence.schema-generation.scripts.action javax.persistence.schema-generation.create-source javax.persistence.schema-generation.drop-source javax.persistence.schema-generation.create-target javax.persistence.schema-generation.drop-target javax.persistence.schema-generation.create-database-schemas javax.persistence.schema-generation.connection	
javax.persistence.lock.timeout javax.persistence.query.timeout	Controlling database timeouts
javax.persistence.validation.mode javax.persistence.validation.group-pre-persist javax.persistence.validation.group-pre-update javax.persistence.validation.group-pre-remove javax.persistence.validation.factory	Controlling Managed Type bean validation, and integration points for javax.validation implementations
javax.persistence.sharedCache.mode javax.persistence.cache.retrieveMode javax.persistence.cache.storeMode	Controlling the second level cache
javax.persistence.bean.manager	Integration with CDI containers
javax.persistence.provider	Used to Select a particular persistence provider implementation This property can be used to bind the persistence unit to a particular provider implementation. If no PersistenceProvider implementation service is available with a matching name then an Exception must be thrown, otherwise the named PersistenceProvider must be used to create the EntityManagerFactory. If an existing EntityManagerFactory existed then it must be destroyed and the new one registered.
javax.persistence.transactionType	Used to control the transaction type of the persistence unit.
javax.persistence.jtaDataSource	If supplied as Strings then these should be

javax.persistence.nonJtaDataSource	passed used in a JNDI lookup. If passed as DataSource objects then these objects should be directly used as sources of connections in the Persistence Unit

5.2 Improvements to the JPA Service

In addition to improving the OSGi JPA service's support by increasing the supported version of the JPA service specification, this RFC proposes a number of other enhancements to the JPA Service.

5.2.1 The JPA Service Extender

The OSGi JPA Service version 1.0 defines an extender model using the Meta-Persistence header, however there is no associated requirement/capability for this extender. This means that automatic resolution and deployment of systems using the JPA service is difficult. This RFC proposes an extender capability that must be defined by the JPA Service implementation.

```
Provide-Capability: osgi.extender;osgi.extender=osgi.jpa;version:Version=1.1
```

The extender capability can be required by persistence bundles to ensure that a JPA service implementation is provisioned automatically

5.2.2 Incomplete/Complete Persistence Units

The OSGi JPA service does not require that a persistence unit be complete (i.e. contain the full configuration for the persistence unit (see JPA service spec 127.4.7)). An incomplete persistence unit is one that does not provide database connection information. Additional configuration and/or configuration overrides can be provided using the EntityManagerFactoryBuilder service.

The lifecycle of incomplete Persistence units is poorly defined in the current specification. For example if an incomplete persistence unit is detected it is unclear whether an EntityManagerFactory should be registered. As this service is unusable without the database connection it should clearly not be available.

This RFC proposes a clarification of the current lifecycle rules

- An EntityManagerFactoryBuilder service must always be registered for complete and incomplete persistence units
- An EntityManagerFactory service must only be registered for a complete persistence unit
- Once configured using an EntityManagerFactoryBuilder service the JPA Service must register the created EntityManagerFactory as a service, adding any supplied configuration properties that match the recommended OSGi service property types as service properties for the EntityManagerFactory service. The javax.persistence.jdbc.password property must be omitted from service properties.
- If the EntityManagerFactoryBuilder service is used to change the configuration being used then any registered EntityManagerFactory service must be unregistered and the new object registered.

6 Data Transfer Objects

7 Javadoc

No New types are proposed for the API.

8 Considered Alternatives

8.1 JPA API Contract

Note that this was removed due to a clarification in the portable-java-contract-definitions which previously disagreed with the spec

Whilst contracts work well for clients of APIs, they do not work well for providers of APIs. As a provider of an API I would typically have an import version range of the form “[X.Y, X.Y+1)”. This ensures that an implementation does not wire to a newer version of the API which may contain methods that the implementation does not. Unfortunately this means that the implementation provider cannot use a contract to import the API, and can only export it. This is because the contracts are separate capabilities, and so there is no way to to write a requirement which rejects a bundle which also offers a higher version of a contract than the implementation requires.

This restriction isn’t always a problem, however in the case of the JPA service both the JPA provider and the JPA Service implementation must implement types from the JPA API. In order for these bundles to talk to one another they must also share a class space for the JPA API. When using the JavaJPA contract this is not possible, as both the JPA Service and the JPA provider must export the API and cannot import it reliably.

To solve this issue it is recommended that the advice for osgi.contract providers be changed. Rather than providing separate contracts for each version of the API that is supported it is proposed that a single contract be provided which contains a List<Version> of all the supported versions:

- osgi.contract;osgi.contract=JavaJPA;version:List<Version>=”2.1,2,1”;uses:=”javax.persistence, javax.persistence.criteria, javax.persistence.metamodel, javax.persistence.spi”

The clients can use this contract unchanged, but the providers can also use the contract with a standard filter range to avoid future incompatible versions of the API.

- osgi.contract;filter:=(&(osgi.contract=JavaJPA)(version>=2.1)(!(version>=2.1.1))”;

This filter ensures that the implementation only wires to a provider of the JPA 2.1 contract and not to a contract which supports any higher version. By using this filter to ensure substitutability, the JPA service implementation and the JPA provider implementation can share a class space for the JPA API.

9 Security Considerations

No new security considerations are required by this design

10 Document Support

10.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. Portable Java Contract Definitions https://www.osgi.org/portable-java-contract-definitions/

[4]. JPA 2.1 Specification http://download.oracle.com/otn-pub/jcp/persistence-2_1-fr-eval-spec/JavaPersistence.pdf

10.2 Author’s Address

Name	Tim Ward
Company	Paremus
Address	
Voice	
e-mail	Tim.ward@paremus.com

10.3 Acronyms and Abbreviations

10.4 End of Document