RFC 215 - Object Conversion

Draft

15 Pages

Abstract

Java is a type safe language that can be used to create applications that are easy to navigate in an IDE and that significantly reduce time to write tests. However, there is a tendency in Java to bypass the type system because it is often deemed easier to use strings instead of proper types: logging, JAX-RS, configuration, records, etc. This RFP investigates the issues that surrounding the use of type safe interfaces and DTOs where traditionally properties and other string based solutions are used.

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 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	01/10/15	Initial version, from RFP, with some initial API proposals.

1 Introduction

This RFC originates from the OSGi enRoute work. In this project, a number of services were identified, designed and implemented based on their needs for web based applications. This document analyzes the application domain and defines the problem that needs to be solved.

Java is a type safe language that be used to create applications that are easy to navigate in an IDE and that significantly reduce time to write tests. However, there is a tendency in Java to bypass the type system because it is often deemed easier to use strings instead of proper types: logging, JAX-RS, configuration, records, etc. This RFP investigates the issues that surrounding the use of type safe interfaces and DTOs where traditionally properties and other strings are used.

2 Application Domain

This section should be copied from the appropriate RFP(s). It is repeated here so it can be extended while the RFC authors learn more subtle details.

3 Problem Description

Experience shows clearly that leveraging the Java type system more and reducing the use of key constants and DSLs in the code can increase the productivity of developers significantly. Java is an excellent language to act as a specification language, which the huge benefit that it can be executed and is extensively supported by IDEs like Eclipse and Intellij.

The DTO model is already powerful in replacing where properties were used but requires more extensive support to match capabilities in Javascript, but then in a type safe way.

However, moving to a more type safe use of Java requires a powerful and flexible data handling that currently lacks. This RFP therefore is seeking proposals for a service that provides the following services:

- General any-to-any type conversion
- Extension to the DTO model that allows more types to be used in its fields
- Extension to the DTO that provides DTOs with an identity and if applicable comparable.
- DTO support for copying, equals, and diffing
- JSON encoding/decoding

4 Requirements

4.1 General

- G0010 – Provide a service that can convert any object to a given type. The specification must clearly outline what conversions are possible but must at least allow the simple types, maps, collections, and arrays.
- G0020 – Provide a type reference class
- G0030 – It must be possible to specify the destination type with a class, a generic type (Type<T>), or a type reference.
- G0040 – It must be possible to convert Strings to popular Java types like Pattern, File, Date, Java Date/Time, UUID, et al. The specification must clearly define the rules for these classes.
- G0045 – It must be possible to convert EventAdmin Event objects and Service Reference objects to Map<String,Object>
- G0050 – The solution should be usable outside of an OSGi Framework, i.e. in plain Java environment.

4.2 Maps

- M0010 – It must be possible to convert a Map or Dictionary to an interface where the method names are used as keys
- M0020 – It must be possible to convert a DTO+ to a Map<String,Object> and vice versa

4.3 DTOs

- D0005 – It must be possible to assign an identity to a DTO. This shall be referred to as a DTO+.
- D0010 – It must be possible to diff two objects of the same type returning information where the DTO+'s differ and in what way.
- D0020 – Provide a proper deepEquals that assumes DTO+
- D0030 – Provide a way for types to handle conversion from and to strings for non-specified types
- D0040 – Provide a way to set/get fields from a DTO+ through a string path.
- D0050 – Provide a base class for identity DTO+s
- D0060 – Provide a compare function for identity DTOs that have a primary key that is comparable
- D0070 – Provide a way to find out if a DTO+ is complex

- D0080 – Provide a way to find out an object is DTO+
- D0090 – Provide a way to verify that an object is a DTO+ and has no cycles
- D0100 – Provide a deep copy routine for a DTO+
- D0110 – Provide a shallow copy routine for a DTO+

4.4 JSON

- J0010 – Provide a JSON encoder and decoder that uses the conversion rules for the conversion from JSON types to destination types
- J0020 – JSON decoding must be able to provide a value without specifying any type for the destination
- J0030 – The output must be an OutputStream, Appendable, or String
- J0040 – The input must be an InputStream, Readable, or String
- J0050 – It must be possible to pretty print the output
- J0055 – It must be possible to generate canonical, compact output
- J0060 – It must be possible to specify the output character set for a stream
- J0070 – It must be possible to specify if nulls are outputed or not
- J0080 – It must be possible to add hook to the conversions for custom types for encoding and decoding

5 Technical Solution

5.1 Converter Service

This section presents a number of different options to shape the Converter API.

5.1.1 Traditional service

Below a more traditional service API for the Converter service can be found. The Converter is obtained from the OSGi Service Registry.

```
public interface Converter {

    <T> T convertJSON(String json, Class<T> clazz);
```

```
    <T> T convertDictionary(Dictionary<String, Object> m, Class<T> clazz);

    <T> T convertMap(Map<String, Object> m, Class<T> clazz);

    <T> T convertServiceReference(ServiceReference sref, Class<T> clazz);

    <T> T convertSingleValue(String string, Class<T> clazz);

    String convertToJSON(Object obj);

    Dictionary<String, Object> convertToDictionary(Object obj);

    Map<String, Object> convertToMap(Object obj);

}
```

5.1.2 Service with Fluent API

Another option would be to use a fluent API, while this is a slightly more modern approach a fluent API is mostly useful when a series of calls on the object are commonly performed, which is not really the case here. However another benefit of such API is that it limits the number of combination APIs needed if many conversion combinations are to be supported.

The `Converter` service will be obtained from the service registry as normal.

```
public interface Converter {

    Converting convert(Dictionary<String, Object> m);

    Converting convert(Map<String, Object> m);

    Converting convert(ServiceReference sref);

    Converting convertJSON(String json);

    Converting convertSingleValue(String string);
}

public interface Converting {

    <T> T to(Class<T> clazz)

    Dictionary<String, Object> toDictionary();

    Map<String, Object> toMap();

    String toJSON();
}
```

5.1.3 Static class

Another option is a static class like below. This will make usage from a non-OSGi context easier, but not using the services model has disadvantages. E.g. replacing the service with an alternative implementation is not as easy. Providing multiple converter services at the same time is not possible.

```
public class Converter {

    public static <T> T convertDictionary(Dictionary<String, Object> m, Class<T> clazz) { … }

    public static <T> T convertJSON(String json, Class<T> clazz) { … }

    public static <T> T convertMap(Map<String, Object> m, Class<T> clazz) { … }

    public static <T> T convertServiceReference(ServiceReference sref, Class<T> clazz) { … }

    public static <T> T convertSingleValue(String string, Class<T> clazz) { … }

    public static String convertToJSON(Object obj) { … }

    public static Dictionary<String, Object> convertToDictionary(Object obj) { … }

    public static Map<String, Object> convertToMap(Object obj) { … }

}
```

5.2 Conversions

The following conversions will be supported

5.2.1 Single-value data types

In the following table recursive conversions are marked as follows Collections.singleton(»String(*v*)) means that *v* is first converted to a String using the rules described there and then the result is passed to Collections.singleton.

If an a runtime type is the same as the target type no conversion is needed and hence this is not mentioned in this table.

dest v / src ->	String	*Boxed*	*primitive*	*Object*	*primitive[]*	*Boxed[]*	*collection<?>*	`null`
String	*v*	*v*.toString()	String.valueOf(*v*)	*v*.toString()	Arrays.toString(*v*) except for char[]: String.valueOf(*v*)	Arrays.toString(*v*)	*v*.toString()	null
String[]	new String[] {*v*}	new String[] {*v*.toString()}	new String[] {String.valueOf(*v*)}	if String[]: *v* otherwise: new String[] {*v*.toString()}	Arrays.stream(*v*). mapToObj(String::valueOf) .toArray(String[]::new)	Arrays.stream(l). map(String::valueOf). toArray(String[]::new)	*v*.stream().map(String::valueOf). toArray(String[]::new)	new String[]{}
List<String>	Collections. singletonList(*v*)	Collections. singletonList(»String(*v*))	Collections. singletonList(»String(*v*))	Collections. singletonList(»String(*v*))	Arrays.stream(*v*). mapToObj(String::valueOf). collect(toList())	Arrays.stream(*v*). map(String::valueOf). collect(toList())	*v*.stream().map(String::valueOf). collect(toList())	Collections. emptyList()
Set<String>	Collections. singleton(v)	Collections. singleton(»String(*v*))	Collections. singleton(»String(*v*))	Collections. singleton(»String(*v*))	Arrays.stream(*v*). mapToObj(String::valueOf). collect(toSet())	Arrays.stream(*v*). map(String::valueOf). collect(toSet())	*v*.stream().map(String::valueOf). collect(toSet())	Collections. emptySet()
Collection<String>	*pick either list or set*							
int	Integer.parseInt(*v*)	*v*.intValue()	if int: *v* otherwise: (int) *v*	***discuss***: throw IAE? or 0?	if *v*.length == 0: 0 otherwise: »int(*v*[0])	if *v*.length == 0: 0 otherwise: »int(*v*[0])	if *v*.size() == 0: 0 otherwise: »int(*v*.iterator(). next())	0
boolean	Boolean.valueOf(*v*)	if Boolean: *v*.booleanValue() otherwise: »int(*v*) != 0	if boolean: *v* otherwise: »int(*v*) != 0	***discuss***: throw IAE? or false?	if *v*.length == 0: false otherwise: »boolean(*v*[0])	if *v*.length == 0: false otherwise: »boolean(*v*[0])	if *v*.size() == 0: false otherwise: »boolean(*v*. iterator().next())	false
char	*v*.length() > 0 ? *v*.charAt(0) : 0	(char) *v*. *number*Value()	(char) *v*	***discuss:*** *v*.toString(). charAt(0)	if *v*.length == 0: 0 otherwise: »char(*v*[0])	if *v*.length == 0: 0 otherwise: »char(*v*[0])	if *v*.size() == 0: 0 otherwise: »char(*v*.iterator(). next())	0
byte	*v*.getBytes()[0] or 0 if no bytes in array.	(byte) *v*.intValue()	(byte) *v*	***discuss***: ?	if *v*.length == 0: 0 otherwise: »char(*v*[0])	if *v*.length == 0: 0 otherwise: »char(*v*[0])	if *v*.size() == 0: 0 otherwise: »byte(*v*.iterator(). next())	0
short								
float								
double	Double. parseDouble(*v*)	v.doubleValue()	(double) *v*	Double. parseDouble(*v*.toString())	if *v*.length == 0: 0.0 otherwise: »double(*v*[0])	if *v*.length == 0: 0.0 otherwise: »double(*v*[0])	if *v*.size() == 0: 0.0 otherwise: »double(*v*.	0

							iterator().next())	
int[]	new int[] {»int(*v*)}	new int[] {»int(*v*)}	new int[] {»int(*v*)}	new int[] {»int(*v*)}	Arrays.stream(*v*). mapToInt(l -> ((*Boxed*) l). intValue()). toArray()	Arrays.stream(*v*). mapToInt(*Boxed*::intValue). toArray();	*v*.stream(). mapToInt(x »int(x)). toArray()	new int[]{}
List<Integer>	Collections. singletonList(»int(*v*));	Collections. singletonList(»int(*v*));	Collections. singletonList(»int(*v*));	Collections. singletonList(»int(*v*));	Arrays.stream(*v*). mapToObj(*Boxed*::valueOf). collect(toList());	Arrays.stream(*v*). map(*Boxed*::intValue). collect(toList());	*v*.stream(). map(x »int(x)). collect(toList());	Collections. emptyList()
Boolean	Boolean.valueOf(*v*)	if Boolean: *v* otherwise: »int(*v*) != 0	if boolean: Boolean.valueOf(*v)* otherwise: »int(*v*) != 0	***discuss***: throw IAE? or false?	if *v*.length == 0: FALSE otherwise: »Boolean(*v*[0])	if *v*.length == 0: FALSE otherwise: »Boolean(*v*[0])	if *v*.size() == 0: FALSE otherwise: »Boolean(*v.* iterator().next())	null
other Boxed types								

5.2.1.1 Object[]

Object[] is similar to Collection<?> although String[] can be converted to List<String> via Arrays.asList(v).

5.2.1.2 Enumerated types

Converting to/from Enum Types is only possible between the enumerated types and their String representation.

5.2.1.3 Other types

Do we want to support the following types: Class, Annotation, BigDecimal/BigInteger?

5.2.2 Complex data structures

Complex data structures hold values of various types. The canonical representation of a complex data structure is a Map. For each supported complex structure a description is made how they are converted to and from the Map representation. Implementations may decide to optimize behavior by providing more direct conversions.

5.2.2.1 Map

Map is the canonical type so no further conversion is needed.

5.2.2.2 Dictionary

A Dictionary is converted to a Map by creating a new map with the exact same key and value pairs.

5.2.2.3 Service Reference

A service reference is converted to a map by taking each property key except ones starting with "." and putting this key in the map. Values are converted using the rules described in section *x* above (TODO OO crashes when I put in an xref). Type+ types such as `String+` or `Integer+` are always converted into a `List` of the given type.

5.2.2.4 Interface

5.2.2.5 DTO

5.2.2.6 JSON

5.2.2.7 … anything else? ...

6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

7 Javadoc

Please include Javadoc of any new APIs here, once the design has matured. Instructions on how to export Javadoc for inclusion in the RFC can be found here: https://www.osgi.org/members/RFC/Javadoc

8 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

9 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

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

Add references simply by adding new items. You can then cross-refer to them by chosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph. ***STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.***

10.2 Author's Address

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drezery
Voice	+33 467542167
e-mail	Peter.kriens@aQute.biz

Name	David Bosschaert
Company	Adobe Systems
Address	
Voice	
e-mail	bosschae@adobe.com

Name	
Company	
Address	
Voice	
e-mail	

10.3 Acronyms and Abbreviations

10.4 End of Document