RFC 215 - Object Conversion

Draft

18 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.

Copyright © OSGi Alliance 2016

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,

Copyright © OSGi Alliance 2016 All Rights Reserved

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

Copyright © OSGi Alliance 2016 All Rights Reserved

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.
0.1	January, 2016	David Bosschaert, changes from Chicago F2F feedback.

Copyright © OSGi Alliance 2016 All Rights Reserved

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

Today, many programs directly interface directly with the outside world through REST, HTTP, and other protocols that are frequently require conversion from strings or byte streams. Data conversion is an inherent part of writing software in a type safe language. In Java, converting strings to proper types or to convert one type to a more convenient type is often done manually. Any errors are then handled inline.

A common problem is interacting with Javascript. Since Javascript has no user defined types, it can get away with relatively clean looking code; the same code in Java usually requires significantly more code due to the required conversions to the fine grained types of Java. This code bypasses the built-in facilities like fields and methods and instead defines constant key strings and embeds the knowledge of the types in a piece of code instead of relying on a central declaration that is then verified by the compiler.

In release 6, the OSGi Alliance introduced Data Transfer Objects (*DTOs*). DTOs are public objects without generics that only contain public fields based on simple types, arrays, and collections. In many ways DTOs replace Java beans. Java beans are hiding their fields and provide access methods but that separated the contract (the public interface) from the internal usage. Though this model has advantages in technical applications (many types, few fields) it tend to be a large overhead when there are relatively few types with lots of fields. i.e. the more common web applications. DTOs unify the specification with the data since the data is what is already public when it is sent to another process or serialized.

By limiting the allowed data types in DTOs and ensuring they have no cycles they can be easily (de)serialized using JSON, providing easy interactions with Javascript.

In enRoute, a *DTO+* is a DTO but it additionally allows many additional types and defines the rules for creating these types in a conversion.

Copyright © OSGi Alliance 2016 All Rights Reserved

In applications, a DTO provides the same role as a Javascript hash/object. In java, however, the fields are typed, providing type checks and content assist in the browser. However, there are similar needs to what Javascript provides when objects are used that way:

- Deep copy – Create two DTOs that are equal but do not share any instances
- Deep equals – Compare two DTOs for equality
- Shallow copy – Create a new DTO but share the fields
- Diff – Calculate the difference between 2 DTOs, providing where and why the the objects are different.
- Path based access – Provide a path based access into a DTO. E.g. a path can be `foo.4.abc`

Several libraries exist which provide a similar form of type conversion.

2.1.1 Commons-Convert

"Commons-Convert is a library dedicated to the task of converting an object of one type to another. The library is ideal for scripting languages or any application that needs to convert (or coerce) one Java object type to another." (http://commons.apache.org/sandbox/commons-convert)

2.1.2 Google Guava

Google Guava (https://github.com/google/guava) Converter API (http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/base/Converter.html)

2.1.3 Dozer

"Dozer is a Java Bean to Java Bean mapper that recursively copies data from one object to another. Typically, these Java Beans will be of different complex types." (http://dozer.sourceforge.net)

2.2 Conversions

The Java language has the concept of a *type*. A type is defined by either a class or one of the generic types.

The first level of conversion of a *value* is limited to *simple types*. Simple types are either booleans, characters, numbers, and String. Simple types do not use generics and have no cardinality, they are immutable. For example, converting a `String` to an `int`. In general the developer that is writing the conversion expects a specific type as input and then calls an appropriate method to do the actual conversion: `int integer = Integer.valueOf(string).`

The second level are *cardinal* types:

- *Collection* – An enumeration of zero or more values.
- *Arrays* – An enumeration of zero or more values.
- *Map* – A mapping from one value to another value

2.3 Reflection

Java is a type safe language that provides access to the type information during runtime. This information is quite extensive and includes generic type information. Though it is impossible to know the type parameters are for an

Copyright © OSGi Alliance 2016 All Rights Reserved

object from a generic class, the places where a generic type is used (a call, extending, method arguments, return type) actually do contain the full generic signatures.

Java does not have a built in concept to create a *type reference* for generic types since an instance does not contain the generic information it was compiled with, this information is erased. A common pattern to provide a generic type signature is to create a `TypeReference<T>` class. To create a reference, an inner subclass is created that then encodes the T in its generic signature for the super relation:

```
new TypeReference<List<Map<Pair<Integer,String>,String>>() {}
```

The TypeReference class then can inspect this information and provides the desired type information with a `Type getType()` method.

Libraries like the bnd Converter can use the reflective information to create another object of a desired type. For example:

```
byte[] barray = Converter.cnv( byte[].class, "1"); // new byte[]{1}

List<Short> shorts = Converter.cnv(

      new TypeReference<List<Short>>(){}, new String[]{"1","2"}); // [1, 2]

FooConfig fooConfig = Converter.cnv( FooConfig.class, map );

Map<String,Object> map = Converter.cnv(

      new TypeReference<Map<String,Object>>(){}, dto );
```

The bnd JSON Codec extend this model to *JSON*. JSON is a syntax to transfer data with only a limited set of types: string, booleans, numbers, arrays, and maps. In general, it is straightforward to map a DTO+ to JSON stream since Java has so much more type information than is required. However, the bnd JSON Codec can take an input stream and a DTO+ and map the JSON input stream to the fields and types defined in the DTO, recursively. Since these types contain the full generic information it is possible to support quite rich DTO+ objects.

2.4 Terminology + Abbreviations

- DTO+ – a DTO with an identity.

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

Copyright © OSGi Alliance 2016 All Rights Reserved

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

Copyright © OSGi Alliance 2016 All Rights Reserved

- 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

Copyright © OSGi Alliance 2016
All Rights Reserved

5 Technical Solution

The solution centers around services to support the conversions: the Converter service which can convert objects from one type to another, and the Encoder service which can encode/decode a specific serialized format.

This RFC also defines a mechanism to use these services from a non-OSGi environment.

5.1 Converter Service

The `Converter` service is used to start a conversion. The service will be obtained from the service registry as normal. The conversion is then completed via the Converting interface that has methods to specify the target type.

```
public interface Converter {

    Converting convert(Object obj);

}

public interface Converting {

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

    <T> T to(TypeReference<T> ref) // to capture the generics of type T

    Object to(Type type) // for use with reflection
}
```

The `TypeReference` class mentioned here is used to obtain Java Generics informaton at runtime. It should be defined in the Converter specification similar to the `TypeReference` class in the OSGi Enroute project:

https://github.com/osgi/osgi.enroute/blob/master/osgi.enroute.base.api/src/osgi/enroute/dto/api/TypeReference.java

TODO Consider use of stream-based approach to generate resulting objects (e.g. create an Event using a Lambda).

5.2 Encoder Service

The Encoder service can be used to encode a given object in a certain representation, for example JSON or XML. The Encoder service can also decode the representation it produced. A single Encoder service can encode/decode only a single format. To support multiple encoding formats register multiple services.

```
public interface Encoder {

    String getName(); // Also service property 'osgi.encoder.name'. E.g. 'My JSON Encoder'
    String getType(); // Also service property 'osgi.encoder.type'. E.g. 'JSON'
    String getVersion(); // Also service property 'osgi.encoder.version'.
```

Copyright © OSGi Alliance 2016 All Rights Reserved

```
    <T> Decoding<T> decode(Class<T> cls);
    <T> Decoding<T> decode(TypeReference<T> ref);
    Decoding<?> decode(Type type);

    Encoding encode(Object obj);
}

public interface Configuration<T> {
    T ignoreNull();
}

public interface Decoding<T> extends Configuration<Decoding<T>> {
    T from(InputStream in);
    T from(InputStream in, String charset);
    T from(Reader in); // TODO do we need this?
    T from(CharSequence in);
}

public interface Encoding extends Configuration<Encoding> {
    Encoding pretty();

    void to(OutputStream out);
    void to(OutputStream out, String charset);
    void to(Appendable out);
    String result();
}
```

The above API can be used it like this:

```
String encoded = encoder.encode(myObj).pretty().ignoreNull().result();
```

5.2.1 Use from outside of OSGi

Use ServiceLoader to find a service of the appropriate type. This can be done with the `Converter` service and the `Encoder` service. The Encoder service has accessors to distinguish one implementation from others.

5.3 Conversions

The following conversions will be supported.

Copyright © OSGi Alliance 2016
All Rights Reserved

5.3.1 Single-value data types and List/Array/Set

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.

The following table is based on the table from the Declarative Service Specification *Coercion from Property Value to Method Type* 112.10 and aims to be backward compatible with that specification.

dest / src	String	Boolean	Character	*Number*	null	empty Collection/Array
String	v	v.toString()	v.toString()	v.toString()	null	“”
boolean	Boolean. parseBoolean(v)	v.booleanValue()	v.charValue() != 0	v.*number*Value() != 0	FALSE	FALSE
char	v.lenght() > 0 v.charAt(0) : 0	v.booleanValue()	v.charValue()	(char) v.*number*Value()	0	0
number	*Number*. parse*Number*(v)	v.booleanValue() ? 1 : 0	(number) v.charValue()	v.*number*Value()	0	0
Class	Bundle.loadClass(v)	throw	throw	throw	null	null
EnumType	*EnumType*. valueOf(v)	throw	throw	throw	null	null
AnnotationType	**Bundle. loadClass(v)**	throw	throw	throw	null	null
BigDecimal/ BigInteger ??						

5.3.1.1 Conversion from non-empty Arrays, Collections to single-value type

The first element is taken and converted into the target element.

Exception: byte[] → String: will be converted by calling new String(byte[] v)

5.3.1.2 Conversion from single value type to Array, List, Set

Conversion to collections:

dest / src	T v
List<T>	Collections. singletonList(v)
Set<T>	Collections. singleton(v)
T[]	new T[] {v};

5.3.1.3 Conversion from Array, List, Set to Array, List, Set

TODO can we support converstions from String[] → List<Integer> or something like this? Do we need special APIs to indicate the target type because of erasure?

5.3.2 Complex data structures

TODO add more detail.

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.

5.3.2.1 Map

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

5.3.2.2 Dictionary

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

5.3.2.3 Interface

In this case support conversion to interfaces that can provide defaults for non-set values:

```
Config {
  int my_value(int defVal);
}

Config cfg = … // created by converter
int val = cfg.my_value(17); // if not set then use 17
```

Copyright © OSGi Alliance 2016
All Rights Reserved

5.3.2.4 Annotation

Just like interface but with the added capability of specifying a default in the annotation definition.

5.3.2.5 DTO

DTOs are classes with public fields and no methods other than the ones provided by the java.lang.Object class. OSGi DTOs extend the org.osgi.dto.DTO class but the converter should ignore this. This is to keep the converter API itself clean from OSGi dependencies.

5.3.3 Adding support to existing OSGi types

Add Map<String,?> getProperties() to various APIs, Event, ServiceReference, BundleContext. This to facilitate converting from those types to other types. It will also keep this API clean of other OSGi deps.

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

Copyright © OSGi Alliance 2016
All Rights Reserved

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.

This section is placed here for the moment, we may use parts of it in the future.

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(v).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*	»int(v.toString())	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	»boolean(v.toString())	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*	»char(v.toString())	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*	»byte(v.toString())	if *v*.length == 0: 0	if *v*.length == 0: 0	if *v*.size() == 0: 0	0

Copyright © OSGi Alliance 2016
All Rights Reserved

					otherwise: »char(*v*[0])	otherwise: »char(*v*[0])	otherwise: »byte(*v*.iterator(). next())	
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*. iterator().next( ))	0
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	»Boolean(v.toString())	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								

Identity conversion (source can be assigned to target → straight passthrought)

Special case for byte[]

ObjectClass::valueOf

String constructor

8.1.1.1 Object[]

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

8.1.1.2 Enumerated types

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

8.1.1.3 Other types

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

Copyright © OSGi Alliance 2016 All Rights Reserved

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

Copyright © OSGi Alliance 2016 All Rights Reserved

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

Copyright © OSGi Alliance 2016
All Rights Reserved