RFC 164 Blueprint Declarative Transaction

Draft

19 Pages

Abstract

Blueprint declarative transaction provides container managed transaction for blueprint beans which enables application developers to focus on the application logic and provides better separation of application and infrastructure code.

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

Namespace Handler – the component of the blueprint container runtime responsible for handling the blueprint transactions namespace. The mechanism by which this is 'plugged in' to the blueprint container is not convered in this specification. This could be blueprint container implementation specific or a standard mechanism may be defined in another specification.

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	*07/12/2010*	*Initial version* *Lin Sun, IBM linsun@us.ibm.com*
0.1	*07/26/2010*	*Address additional comments -* *1. Disallow two *s next to each other* *2. Ability to specify bundle wide transaction.*
0.2	*09/17/2010*	*Simplified the rules to determine the transaction attribute* *Simplified the value for the method and bean attributes.*

Revision	Date	Comments
0.3	*11/16/2010*	*Add: statement for synthetic beans.* *Clarify namespace handler requirement and a few minor errors here and there.* *Mark transaction attribute as optional and default to Required.*
0.31	*04/05/11*	*Graham Charters* *Tidy up for draft publication.*
0.32	*15th June 2012*	*Graham Charters* *Minor corrections to schema and examples*

1 Introduction

RFP 138 was approved which defined requirements for blueprint declarative transaction. This RFC describes the means to use blueprint interceptors and customized blueprint namespace to support blueprint declarative transaction.

2 Application Domain

2.1 Overview

The Blueprint Container specification version 1.0 in OSGi Compendium Release 4.2 defines a dependency injection framework for OSGi bundles that understands the unique dynamic nature of services. Bundles in this programming model contain a number of XML de nition resources which are used by the Blueprint Container to wire the application together and start it when the bundle is active.

The Java Transaction API (JTA) Transaction Services specification in OSGi Enterprise Release 4.2 is based on JTA 1.1 specification and defines the use of JTA through 3 transaction services: User Transaction, Transaction Manager, and Transaction Synchronization.

With these two specifications, a blueprint application developer can obtain the transaction services from the OSGi service registry, set the transaction boundaries and manage the transaction in the application code. However, this is no defined method to enable the developer to declare transactional demarcation for blueprint components and no defined contract that a blueprint container could offer when processing such transactional declarations.

2.1 Sample User Scenario

The blueprint container manages various components such as bean, service, reference-list and reference elements. Blueprint beans are declared using the bean element. The following defines a single bean called TestBeanImpl implemented by the POJO org.apache.aries.simple.TestBeanImpl.

blueprint.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
    <bean id="TestBeanImpl" class="org.apache.aries.simple.TestBeanImpl" />
</blueprint>
```

If a transaction is required in the POJO org.apache.aries.simple.TestBeanImpl, the application developer would currently have to obtain the User Transaction service from OSGi service registry, set the boundaries of transaction and manage the transaction in the application code, for example the insertRow method in TestBeanImpl.java:

TestBeanImpl.java

```
public class TestBeanImpl {

  public TestBeanImpl() {
     ...
    }

  public void insertRow(String name, int value) throws SQLException {
    // get the user transaction service
    UserTransaction ut = getUserTransactionService();

    // begin the user transaction
    ut.begin();

    try {
```

```
        // start the operation to insert name, value into the db
        ...
      } finally {
        // commit the user transaction
        ut.commit();
      }
    }
}
```

This form of transaction management can be referred to as “application managed transactions” or “bean managed transactions” , whereby the bean code explicitly manages the boundaries of the transaction. Given that the primary purpose of a container is to manage the lifecycle and qualities of service of the entities it contains, it is desirable to de ne a standard mechanism for container to manage transactions on behalf of those entities. This enables the bean developer to focus on the application logic and provides better separation of application and infrastructure code.

A useful model to refer to that is extremely familiar to Java EE developers is the Java EE enterprise bean with container managed transactions, where the Enterprise Java Bean (EJB) container manages the boundaries of the transactions. The vast majority of enterprise beans use container-managed rather than bean-managed transactions.

3 Problem Description

It is possible that a transaction is required in one or more methods of the blueprint managed components de ned in blueprint XML de nition. The blueprint container speci cation does not describe a means to allow application developers to de ne transaction attributes (such as Required, RequiresNew, NotSupported, Supports, Mandatory and Never, as provided by the EJB container) in the blueprint XML de nition. Instead, if an application developer has chosen to use the blueprint container to manage his bundles, the application developer would have to set the transaction boundary and manage the transaction in the application code, whenever a transaction is required. There is no container managed transaction de ned for blueprint components.

This proposal is to provide container managed transaction for blueprint managed components, where application developers can use blueprint XML de nition to declare transactions and their associated transaction attributes within one or more methods of the blueprint container managed components, so that the transaction can be managed by the container automatically based on the speci ed transaction attributes and the bean methods to which they apply.

4 Requirements

BDT01 - Blueprint Declarative Transaction must support these types of Transaction attributes: (Required, RequiresNew, NotSupported, Mandatory, Supports and Never).

BDT02 - Blueprint Declarative Transaction must support configuration at the blueprint bean level.

BDT03 - Blueprint Declarative Transaction must support configuration at the bean method level.

BDT04 - Blueprint Declarative Transaction must support configuration of a default transaction attribute that can be overridden by more specific declarations.

BDT05 - Blueprint Declarative Transaction must define the container's behavior for completing transactions in the presence of a normal successful method execution or exception.

BDT06 – Blueprint declarative transaction must be configurable in blueprint XML definition file.

BDT07 - It MUST be possible to implement Blueprint Declarative Transactions using Blueprint Interceptors.

BDT08 - Blueprint Declarative Transactions MUST define a custom namespace.

5 Technical Solution

5.1 Related RFCs

RFC 155 Blueprint namespace enables user defined namespace using an XML schema file. In this RFC, a transaction custom namespace is introduced to allow users to easily declare transaction attribute within one or more methods of the blueprint bean component. RFC 166 blueprint bean interceptor can be used to allow blueprint container to intercept method invocation to add additional transactional demarcation operations before and after the method invocation.

This RFC and subsequent specification should be implementable using the two aforementioned RFC, but are not required to be.

5.2 Blueprint Transaction XML Schema

By leveraging RFC 155, the following blueprint transaction XML schema is proposed for blueprint transaction:

```
<?xml version="1.0" encoding="UTF-8">
<xsd:schema xmlns="http://www.osgi.org/xmlns/blueprint/transactions/v1.0.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.osgi.org/xmlns/blueprint/transactions/v1.0.0"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="1.0.0">

  <xsd:annotation>
    <xsd:documentation>
      <![CDATA[
        This is the XML Schema for the OSGi Blueprint declarative
transaction 1.0.0 development descriptor.  Blueprint declarative transaction
is a custom namespace for OSGi Blueprint service 1.0.0 development
descriptor.  It is designed to decorate transaction attribute of the bean
components, which can be done at the bean level or at the bundle wide level.
In other words, the transaction element can reside in the bean element or in
the root of the blueprint element as a top level transaction element.
Blueprint configuration files using this schema must indicate the schema
using the transactions/v1.0.0 namespace.  For example,
        <transaction
xmlns="http://www.osgi.org/xmlns/blueprint/transactions/v1.0.0">
        if used as a qualified namespace, "tx" is the recommended namespace
prefix.
      ]]>
    </xsd:documentation>
  </xsd:annotation>

  <xsd:simpleType name="TtransactionAttribute">
    <xsd:annotation>
      <xsd:documentation>
        <![CDATA[
          The TtransactionAttribute type defines the transaction attribute
```

```
for blueprint declarative transaction.
        ]]>
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Required" />
      <xsd:enumeration value="Mandatory" />
      <xsd:enumeration value="RequiresNew" />
      <xsd:enumeration value="Supports" />
      <xsd:enumeration value="NotSupported" />
      <xsd:enumeration value="Never" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="Ttransaction">
    <xsd:annotation>
      <xsd:documentation>
        <![CDATA[
          Ttransaction defines one or more methods that are intercepted with
the specified transaction attribute. Multiple methods names can be wild-
carded with a '*'.  Method can be a mixture of fixed string and one wild-
card.   Methods can be whitespace or comma separated. The bean attribute
refers to the bean component id and can only be used for top level
transaction element.  Similar as methods, beans can be whitespace separated
and be a mixture of fixed string and one wild-card.
        ]]>
      </xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="method" type="xsd:string" />
      <xsd:attribute name="value" type="TtransactionAttribute"
use="optional" default="Required" />
      <xsd:attribute name="bean" type="xsd:string" />

  </xsd:complexType>

  <xsd:element name="transaction" type="Ttransaction">
    <xsd:annotation>
      <xsd:documentation>
        <![CDATA[
          The <transaction> element is the root element for blueprint
declarative transaction
        ]]>
      </xsd:documentation>
```

```
    </xsd:annotation>
  </xsd:element>

</xsd:schema>
```

When the tx element is not specified in the blueprint XML definition file, the default behavior is equivalent to:

```
<tx:transaction value="NotSupported" />
```

5.2.1 The Transaction Element

The transaction element can be used as a child of the bean element to specify the bean level transaction configuration for the bean. The transaction element can also be used at the top level (e.g. at the root of the blueprint element) to specify the bundle wide transaction for the blueprint managed bundle. The following must be used from high priority to low priority to determine which transaction attributes apply to a particular method in a bean component:

1. bean level transaction element with the method attribute.
2. bean level transaction element without the method attribute.
3. top level transaction element with both the bean attribute and method attribute
4. top level transaction element with only the bean attribute
5. top level transaction element with only the method attribute
6. top level transaction elements with no bean or method attribute specified

When using the above selection rules, if the bean attribute or method attribute is set to '*', it must be treated as the attribute is not specified. The top level transaction elements should not apply to synthetic beans.

If there are potential multiple matches for bean level transaction, the transaction namespace handler must throw an Exception back to the Blueprint Container to fail the creation of the Blueprint Container. If there are multiple matches for bundle wide transaction, the transaction namespace handler must throw an Exceptionback to the Blueprint Container to fail the creation of the Blueprint Container. The Blueprint FAILURE event must be emitted as the result of the failure, which is specified in the Blueprint Container Specification.

5.2.2 The Method Attribute

The method attribute is used to specify one or more method names and method names can be wild-carded with one '*'. '*' can appear anywhere in the name and can only appear once. Methods can be listed (whitespace or comma separated) and be a mixture of fixed string and one wild-card. If there are potential multiple matches, the transaction namespace handler must throw an Exception back to the Blueprint Container to fail the creation of the Blueprint Container.

For example, when the transaction namespace handler parses the 2nd transaction element inside the bean, it detects multiple matching for potential methods like count1Row thus the namespace handler must throw a Throwable back to the Blueprint Container to fail the creation of the Blueprint Container for the bundle.

```
<bean ...>
  <tx:transaction method="count*" value="Required" />
  <tx:transaction method="count*Row" value="RequiresNew" />
</bean>
```

The method attribute is optional. When the method attribute is not being specified, , it is equivalent to method attribute set to '*'. For example:

```
<bean ...>
    <tx:transaction method="*" value="Required" />
</bean>
```

5.2.3 The Transaction Attribute

The transaction attributes(Required, Mandatory, RequiresNew, Supports, NotSupported, and Never) are defined in Enterprise Java Bean3.0 specification section 136.6.2. The Blueprint container will manage transaction boundaries in the same manner defined for the EJB container, which is not repeated here. After parsing the blueprint XML definition file, the blueprint container must demarcate transactions based on the transaction attribute specified in the blueprint XML definition file.

5.2.4 The Bean Attribute

The bean attribute can only be specified in top level transaction element. It is used to specify one or more bean component ids and beans can be wild-carded with one '*'. '*' can appear anywhere in the name and can only appear once. Beans can be listed (whitespace or comma separated) and be a mixture of fixed string and wild-card.

The bean attribute is optional. When the bean attribute is not specified, the default value is an asterisk character (*). For example:

```
<blueprint>
    <tx:transaction method="Insert*" value="Required" />
</blueprint>
```

The blueprint namespace handler design must provide a way to allow the transaction namespace handler to perform post blueprint namespace processing. This ensures all blueprint components are registered in the Component Definition Registry when the transaction namespace handler parses the bundle wide transaction elements and uses the Component Definition Registry to validate the bean attribute. .

5.2.5 Transaction Examples At Bean Level

Transactions can be configured at the bean level. You must specify the value attribute in the transaction element. Valid values are those that are defined by Java EE, that

is, Required, RequiresNew, NotSupported, Mandatory, Supports or Never. For example:

```
<bean ...>
    <tx:transaction method="updateOrder" value="Required" />
</bean>
```

You can also use the wildcard character anywhere in a method name and but you can only use it once. For example:

```
<bean ...>
    <tx:transaction method="update*Ord" value="Required" />
</bean>
```

Methods can be listed (whitespace or comma separated) and be a mixture of fixed string and wild-cards, for example,

```
<bean ...>
    <tx:transaction method="update* makeItSo" value="Required" />
</bean>
```

Multiple method configurations can appear in the same component, for example

```
<bean ...>
    <tx:transaction method="update* makeItSo" value="Required" />
    <tx:transaction method="recordStatus" value="RequiresNew" />
</bean>
```

Method configurations can appear alongside one component configuration, where the methods config takes precedence.

```
<bean ...>
    <tx:transaction value="Required" />
    <tx:transaction method="recordStatus" value="RequiresNew" />
</bean>
```

Wildcard matching and selection behavior is determined by the rules specified in section 5.1.2.

5.2.6 Transaction Examples At Bundle Wide Level

Transactions can be configured at the bundle wide level. Same as bean level configuration, you must specify the value attribute in the transaction element, for example:

```
<blueprint>
    <tx:transaction value="Required" />
    ...
</blueprint>
```

You can also use the wildcard character anywhere in a method name or a bean name but you can use the wildcard character only once. For example:

```
<blueprint>
    <tx:transaction bean="bean1*" method="update*Ord" value="Required" />
    ...
</blueprint>
```

Methods or Beans can be listed (whitespace or comma separated) and be a mixture of fixed string and wild-cards, for example,

```
<blueprint>
    <tx:transaction bean="bean1* bean2*" method="update* makeItSo"
value="Required" />
    ...
</blueprint>
```

Multiple configurations can appear in the same bundle, for example

```
<blueprint>
    <tx:transaction method="update* makeItSo" value="Required" />
    <tx:transaction bean="bean1" bean="update*" value="RequiresNew" />
</blueprint>
```

When multiple configurations exist, the selection behavior is determined by the rules in section 5.2.1.

5.2.7 Transaction Examples At Bean Level And Bundle Wide Level

Transaction can be configured at bean level, bundle wide level or both.

```
<blueprint>

<tx:transaction value="Required" />

<tx:transaction method="get*" value="Supports" />

<tx:transaction bean="noTx" value="Never" />

<tx:transaction bean="someTx" method="get*" value="Mandatory" />


<bean id="requiresNew" class="foo">
  <tx:transaction value="RequiresNew"/>
</bean>
```

```
<bean id="noTx" class="foo"/>

<bean id="someTx" class="foo"/>

<bean id="anotherBean" class="foo"/>

<blueprint/>
```

In the above example the following would apply:

- For the bean requiresNew, all methods have REQUIRESNEW behaviour.
- For the bean noTx, all methods have NEVER behaviour.
- For the bean someTx, methods starting with "get" are MANDATORY, and all other methods are REQUIRED.
- For the bean anotherBean, all methods that start with "get" are *SUPPORTS*, and all other methods are REQUIRED.

5.3 Declaring a Dependency on Blueprint Transactions

As with extenders, a blueprint bundle that uses transactions is able to declare a dependency on a bundle that provides that capability. A provider of blueprint transactions support must delcare that it provides this capability. This is done using the Require-Capability and Provide-Capability Headers, respectively.

The header a bundle uses to state that it provides the blueprint transactions capability is:

```
Provide-Capability: osgi.extender; osgi.extender="osgi.blueprint.transactions";
 uses:="org.osgi.service.blueprint";version:Version="1.0"
```

The uses constraint is intended to ensure class-space consistency between the blueprint extender, blueprint extendee and blueprint transactions bundles. The blueprint transactions bundle must import the org.osgi.service.blueprint package and also ensure it is wired to the correct blueprint extender. The latter can be achieved by importing a package specific to the blueprint extender (e.g. its proprietary API for namespace extensions) or through Require-Bundle.

The header a bundle may use to state that it requires the blueprint transactions capability is:

Require-Capability: osgi.extender; filter="(osgi.extender=osgi.blueprint.transactions)"

Note, adding the supported namespaces to the osgi.blueprint extender capability was considered, but this would require the namespaces to be provided by the core blueprint extender, rather than a separate bundle.

5.4 Sample User Scenario

Continuing to use the example in the section 2, the following defines the single bean called TestBeanImpl, using the declarative transaction on the method insertRow with transaction attribute value required.

blueprint.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
      xmlns:tx="http://www.osgi.org/xmlns/blueprint/transactions/v1.0.0">
    <bean id="TestBeanImpl" class="org.apache.aries.simple.TestBeanImpl">
      <tx:transaction method="insertRow" value="required"/>
    </bean>

</blueprint>
```

The TestBeanImpl.java example used in section 2 can be simplified as below:

TestBeanImpl.java

```
public class TestBeanImpl {

   public TestBeanImpl() {
      ...
     }

   public void insertRow(String name, int value) throws SQLException {
     // start the operation to insert name, value into the db
     ...
   }
}
```

Based on the application's requirement, application developers can set different transaction attributes on specific methods, such as Required, RequiresNew, Mandatory, NotSupported, Supports, and Never.

This simplifies greatly the transaction management in application code TestBeanImpl.java.

~~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.Initial Spec Chapter~~

~~Provide a link to where the Initial Spec Chapter can be found. The Initial Spec Chapter is typically written by someone other than the author(s) of this RFC and represents a rewrite of this document as close as possible to what will ultimately appear in the OSGi Specifications. It will be used by the Specification Editor as the basis for the ultimate specification chapter.~~

~~*The spec template and writing guidelines can be found here:*~~

~~*https://www.osgi.org/members/svn/documents/trunk/templates/specification-template-oo.ott*~~

~~*https://www.osgi.org/members/svn/documents/trunk/templates/specwriting.pdf*~~

~~*JMX API*~~

~~*RFC 139 describes the JMX API to the OSGi Framework and a number of standard OSGi Services. The JMX specification is also present as chapter 124 in the OSGi spec documents.*~~

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

~~*The JMX API for the design in this RFC should be described here and if there is no JMX API an explanation should be given explaining why this is not applicable in this case.*~~

~~*None*~~

~~Command Line API~~

~~*None*~~

6 Security Considerations

~~*Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.*~~None identified.

7 Document Support

7.1 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]. EJB specification v3.0 - http://java.sun.com/products/ejb/

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.

7.1 Author's Address

Name	Lin Sun
Company	IBM
Address	3039 Cornwallis Road, RTP, NC, 27560
Voice	919-254-6388
e-mail	linsun@us.ibm.com

Name	Graham Charters
Company	IBM
Address	IBM UK Ltd, MP211, Hursley Park Road, Winchester, SO21 2JN
Voice	+44 1962 816527
e-mail	charters@uk.ibm.com

Name	Ian Robinson
Company	IBM
Address	IBM UK Ltd, MP211, Hursley Park Road, Winchester, SO21 2JN
Voice	+44 1962 818626
e-mail	ian_robinson@uk.ibm.com

7.2 Acronyms and Abbreviations

7.3 End of Document