Blueprint Container 1.1

Draft

44 Pages

Abstract

Update Blueprint Container to 1.1 with some features requested via bug reports or directly from users.

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 Document Information **2**
- 0.1 License 2
- 0.2 Trademarks 3
- 0.3 Feedback 3
- 0.4 Table of Contents 3
- 0.5 Terminology and Document Conventions 4
- 0.6 Revision History 4

1 Introduction **5**

2 Application Domain **5**

3 Problem Description **6**
- 3.1 3.1 Non-Damped Reference Managers 6
- 3.2 3.2 Blueprint Grace Timeout 6
- 3.3 3.3 Bug 2233 –'Satisfied' Lifecycle Notification 6
- 3.4 3.4 Injection of Service Properties 6
- 3.5 3.5 Bug 2192 - Damping of Factory Services 7
- 3.6 3.6 Bug 1295 - Allow Namespace handlers to use inline Blueprint elements 7
- 3.7 3.7 Bug 2406 – Add blueprint extender capability definition 7
- 3.8 Bug 2484 – Address lack of Blueprint opt-in header 7
- 3.9 3.8 Allow service properties to be added via <property> elements in Service Manager 8

3.10 3.9 Add synchronous start mode for Blueprint Container....8

4 Requirements....8

5 Technical Solution....9
5.1 Non-Damped Reference Managers....9
5.2 Blueprint Grace Period enhancements....9
5.3 New Satisfied Life-cycleEvent....10
5.4 Injection of Service Properties....10
5.5 Factory Services....10
5.6 Inlining of Blueprint elements....10
5.7 Extender Capability....10
5.8 Allow <property> elements in Service Managers....11
5.9 Add synchronous start mode for Blueprint Container....11
5.10 Blueprint Extender Configuration (Bug 2484)....11
5.10.1 Extendee Header Behavior....11
5.10.2 TODO: Other things configured through the dictionary....12

6 Considered Alternatives....12

7 Security Considerations....12

8 Document Support....12
8.1 References....12
8.2 Author's Address....13
8.3 Acronyms and Abbreviations....13
8.4 End of Document....13

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	*21/05/12*	*Initial Draft Requirements*
0.2	*2nd November 2012*	*First stab at design.*
0.3	*27th February*	*Updates from Nov 2f2 & tidy-up for draft publication.*
0.4	*09th August 2013*	*Updated to include default service implementations*

Revision	Date	Comments
0.5	*11th September*	*Updates for RFC0195.*
0.6	*08th October*	*Updates from Septembers OSGi F2F*
0.61	*21st January*	*Added Blueprint 1.1 Schema and updates from Nov call.*

1 Introduction

This RFC will propose some new minor feature enhancements to the existing 1.0 Blueprint Container specification. The feature requests have come from bug reports or through user experience.

2 Application Domain

The following was taken from the Blueprint 1.0 requirements and design documents (RFP 76 & RFC 124. respectively)

The primary domain addressed by this RFP is enterprise Java applications, though a solution to the requirements raised by the RFP should also prove useful in other domains. Examples of such applications include internet web applications providing contact points between the general public and a business or organization (for example, online stores, flight tracking, internet banking etc.), corporate intranet applications (customer-relationship management, inventory etc.), standalone applications (not web-based) such as processing stock feeds and financial data, and “frontoffice” applications (desktop trading etc.). The main focus is on server-side applications.

The enterprise Java marketplace revolves around the Java Platform, Enterprise Edition (formerly known as J2EE) APIs. This includes APIs such as JMS, JPA, EJB, JTA, Java Servlets, JSF, JAX-WS and others. The central component model of JEE is Enterprise JavaBeans (EJBs). In the last few years open source frameworks have become important players in enterprise Java. The Spring Framework is the most widely used component model, and Hibernate the most widely used persistence solution. The combination of Spring and Hibernate is in common use as the basic foundation for building enterprise applications. Other recent developments of note in this space include the EJB 3.0 specification , and the Service Component Architecture project (SCA).

Some core features of the enterprise programming models the market is moving to include:

- A focus on writing business logic in “regular” Java classes that are not required to implement certain APIs or contracts in order to integrate with a container

- Dependency injection: the ability for a component to be “given” its configuration values and references to any collaborators it needs without having to look them up. This keeps the component testable in isolation and reduces environment dependencies. Dependency injection is a special case of Inversion of Control.

- Declarative specification of enterprise services. Transaction and security requirements for example are specified in metadata (typically XML or annotations) keeping the business logic free of such concerns. This also facilitates independent testing of components and reduces environment dependencies.

- Aspects, or aspect-like functionality. The ability to specify in a single place behavior that augments the execution of one *or more* component operations.

In Spring, components are known as “beans” and the Spring container is responsible for instantiating, configuring, assembling, and decorating bean instances. The Spring container that manages beans is known as an “application context”. Spring supports all of the core features described above.

3 Problem Description

3.1 No Timeout~~Non-Damped~~ Reference Managers

~~Blueprint reference managers have mandatory damping, however, not all use cases desire damping.. The timeout period can be changed, but this requires additional configuration and does not allow damping to be turned off for an individual reference. There is a need for non-damped reference managers. One benefit of this would be to avoid a timeout period which would allow Blueprint to immediately flag the component as having missing dependencies should the service not be available.~~

Blueprint reference managers have mandatory damping, and this damping always waits for a certain amount of time for a service to become available before it throws a ServiceUnavailableException. You can alter the timeout to be shorter than the default, but there is always a wait.

3.2 Blueprint Grace Timeout

When the Blueprint grace period is reached, the Blueprint container no longer checks for dependencies, even if outstanding dependencies are then satisfied. At this point, the only way the container can be refreshed is to restart the bundle. There needs to be a better way to control the life-cycle of blueprint containers that does not risk them becoming zombies.

3.3 Bug 2233 –'Satisfied' Lifecycle Notification

The lifecycle events for Blueprint aren't sufficient to get a complete understanding of what Blueprint is currently doing. If blueprint enters the grace period waiting for dependencies, there is no event fired to say when it's dependencies have been satisfied. This is an important piece of information to know when trying to track down blueprint problems.

3.4 Injection of Service Properties

If a User wants to access the service properties of an injected service, they must first be injected with the service reference and then programmatically use the service reference to access the properties. Not only is it an inconvenient extra step to have to do to get to the properties, it also ties the bean to the OSGi APIs, which can impact the ability to unit test the bundles.

3.5 Bug 2192 - Factory Services Lifecycle issues

If a factory Service is removed and replaced by another factory service, Blueprint doesn't recreate the beans that were created with the original factory.

This is a Blueprint example for the problem:

```
<reference id="foo" interface="javax.persistence.EntityManagerFactory/>

<bean id="not_working_after_update" class="SomeClass">
    <property name="bar">
        <bean factory-component="foo" factory-method="createEntityManager"/>
    </property>
</bean>
```

In the example, the bean created by the service Reference foo, should be deleted and re-created, if the EntityManagerFactory service that foo is bound to, is removed and replaced by another EntityManagerFactory service.

3.6 Bug 1295 - Allow Namespace handlers to use inline Blueprint elements

Custom namespace handlers should be able to include / in-line elements from the Blueprint Schema. Bug 1295 was raised against RFC 155 (Namespace Handlers), but requires changes to the core Blueprint specification that are potentially valuable even in the absence of a namespace handler standard.

```
<tx:transacted>
  <bean ... />
</tx:transacted>
```

3.7 Bug 2406 – Add blueprint extender capability definition

The latest Core OSGi specification introduced the standard capability namespace for extender implementations, and the Blueprint extender needs to implement this new namespace e.g.

```
Provide-Capability: osgi.extender; osgi.extender="osgi.blueprint";
 uses:="org.osgi.service.blueprint.container,
 org.osgi.service.blueprint.reflect";version:Version="1.1"
```

3.8 Bug 2484 – Address lack of Blueprint opt-in header

Best practice design for extenders is for them to require an opt-in header specified in the extendee bundle's manifest. Blueprint 1.0, however, treats the Bundle-Blueprint header as an opt-out (when no value is specified). This means the default behavior for Blueprint 1.0 is to search all bundles for blueprint configuration, unless they contain the opt-out header, resulting in significant performance issues for large deployments. Blueprint needs a way to address this, ideally without breaking backwards compatibility.

3.9 Allow service properties to be added via <property> elements in Service Manager

When you want to define service properties within a Service Manager, you have to define a service-properties element, and then define entry elements for each property, which seems unnecessarily verbose and over-complicated.

3.10 Make it easier to consume optional services

The blueprint container allows reference-managers to be “optional”, meaning that they may have no backing service. As identified in 3.1, when a reference manager has no backing implementation it can wait for a long time, then it throws ServiceUnavailableException. Whilst reducing this timeout to zero is helpful in error cases for mandatory services, for optional reference managers “no backing service” is a valid, main-path state. Throwing an exception for every one of these invocations is both wasteful and counter-intuitive – it is not an “exceptional” case.

Currently to avoid this scenario blueprint components must be declared as reference-listener objects. They then receive notifications of the reference bind/unbind events. This adds significant additional threading complexity to the blueprint component, and (assuming it is not safe to hold a lock/monitor while calling the reference) there is still a race condition that can result in a ServiceUnavailableException.

Blueprint's purpose is to make exposing and consuming services simpler. The situation outlined above is not simple, and needs to be improved.

3.11 Ensure Blueprint supports new OSGi Service scopes.

RFC 0195 introduces a new Service Factory type, PrototypeServiceFactory, that can return multiple instances of a particular service to the same consuming bundle. Blueprint needs to support this new RFC0195.

3.12 Allow other value types to be used in Service Properties and Map values.

By default, Service Property values and Map Entry values are Strings. If a non-String value type is required, nested entry elements need to be defined, in order to be able to configure the type of the value. It should be easier to specify the type without needing to add the extra nested elements.

4 Requirements

Blueprint1 – A component MUST be able to request that a Reference Manager ~~is not damped.~~should not wait for a service dependency.

Blueprint2 – It MUST be possible for a Blueprint Container to use the grace period without risking permanent failure (requiring a bundle restart) if the period expires. Once the grace period has been reached, the Blueprint container MAY decide to partially start the beans and services that have had their dependencies satisfied, or it MAY decide to continue to wait for outstanding dependencies. Relevant Lifecycle events MUST be issued for each scenario.

Blueprint3 – The Blueprint Container MUST issue an additional lifecycle event when the dependencies of a component are satisfied.

Blueprint4 – A component MUST be able to have the service's properties injected directly without requiring the bean to use OSGi framework APIs.

Blueprint5 – The Blueprint Container MUST ensure that beans created by factory services are kept in sync with the life-cycle of the factories. For example, if the factory service is replaced, the beans from the old factory should be removed and new beans created using the new factory service.

Blueprint6 – Custom namespaces MUST be able to include or have inlined Blueprint elements from the Blueprint Schema. This MAY mean changing the Blueprint Schema to define the Blueprint element types upfront, and refer to them throughout the Schema structure, rather than, as it does today, in-lining the element type definitions within the Schema structure.

Blueprint7 – The Blueprint Container MUST support the extender capability definition.

Blueprint8 – Blueprint Service Managers MUST support <property> elements.

Blueprint10 – It MUST be possible to configure Blueprint processing such that the extender does not process any bundles that do not contain the Bundle-Blueprint header.

Blueprint11 – It MUST be possible to write a blueprint-managed component with an optional reference without having to be registered as a reference-listener, or handling ServiceUnavailableException

Blueprint12 – The Blueprint Container MUST support the new Service Scope specification.

Blueprint13 – It MUST be easier to define non-String values in Service Properties and Maps.

5 Technical Solution

5.1 Non-Damped Timeout Reference Managers

To allow Reference Managers to be configured to not dampwait for the services that they reference, the existing optional timeout property withll be able to be set to -1, This new value indicates that the reference should not be dampedwait, and should throw a ServiceUnavailableException should the reference not exist at the time the element is processed by the Blueprint container. This attribute is only available on the reference element, and not on the reference-list element, as reference-lists are not damped.

Reference Manager dampingtimeouts can also be specified at the blueprint element level using the new optional `timeout` attribute. The attribute applies to all Reference Managers for the corresponding Blueprint container. This attribute can have the same values as the timeout attribute on the reference Manager, so setting this to -1 would mean all of the Blueprint container's Reference Managers would not dampwait their services.

The timeout attribute on individual reference elements will take precedence over the blueprint element's attribute.

Adding these attributes will require a new version of the Blueprint XML schema.

Note: This requirement has changed as all referenced services are damped, so this requirement is now introducing a 0 ms timeout. As we already have the ability to drop the wait to 1ms, I wonder if this requirement really makes any sense.

5.2 Blueprint Grace Period enhancements

When the grace period is reached, Blueprint will have two new options available in order to avoid creating 'zombie' Blueprints. These are specified on the existing directive called, blueprint.graceperiod which is specified on the bundle symbolic name. The new values are `allowPartial` , `forever`. For clarity, we will also introduce new values `fail` and `none`, which will have the same meaning as true and false respectively.

When a value of `allowPartial` is specified, Blueprint must wait for the graceperiod and when the graceperiod is reached, rather than issuing a FAILURE event and performing “Destroy”, it must create the blueprint container and set up as much of the blueprint as possible, based on the set of satisfied mandatory references in the same way it would if the graceperiod were set to `false`.

When a value of `forever` is specified, Blueprint must wait for the graceperiod and when the graceperiod is reached, rather than issuing a FAILURE event and performing “Destroy”, the Blueprint runtime must issue a new GRACE_PERIOD event and begin a new grace period.

Any blueprint bundles that have the new 1.1 namespace witl have the allowPartial behaviour as default. If a host bundle has fragment(s) and any of these bundles use the 1.1 namespace, then all these related bundles will use allowPartial by default.

We will need a new version of the Blueprint XML schema for this new attribute. to indicate the change in behaviour is required.

5.3 New Satisfied Life-cycleEvent

When the Blueprint Container processes service references and issues a GRACE_PERIOD event because there are missing mandatory dependencies, it will now issue a SATISFIED event when all mandatory dependencies are finally satisified and before proceeding to the “Register Services” process.

The Blueprint Event property DEPENDENCIES will have the same array of Strings containing the dependencies that was issued in the corresponding GRACE_PERIOD event.

5.4 Injection of Service Properties

When looking for an appropriate bean set method to call for injection of a service object, as a last option, Blueprint will now also look for a method with the following signature:

set{PropertyName}(T ref, java.util.Map<String, ?> props)

PropertyName is the name of the bean property, specified in the property element, for example

```
public class C {
  public void setProxy(T ref, Map<String, ?> props) { ... }
}
<reference id="p" interface="T"/>
<bean id="c" class="C">
  <property name="proxy" ref="p"/>
</bean>
```

It is important to note that the injected service properties are an unmodifiable map of properties, and is a snapshot at the time the map is injected.

5.5 Factory Services

Currently when factory services are used, the actual bean created by the factory service, is injected. If the factory service is replaced by another factory service, the blueprint container doesn't replace the existing beans with ones created using this new factory.

The Blueprint container will ensure that whenever a factory service is replaced, all beans created by the original factory service are removed and replaced by new beans created by the new service. The factory service will also now return a damped proxy, rather than the actual bean.

5.6 Inlining of Blueprint elements

In order to support the ability for Custom Namespace Handlers to use inlined Blueprint elements,the Blueprint schema needs to be amended to declare the element types outside of the nested groups and reference them within the groups e.g.

```
<xsd:group name="allComponents">
<xsd:choice>
<xsd:element ref="service"/>
<xsd:element ref="ref-list"/>
<xsd:element ref="ref-set"/>
<xsd:group ref="targetComponent"/>
</xsd:choice>
</xsd:group>
<xsd:element name="service" type="Tservice"/>
<xsd:element name="ref-list" type="Tref-collection"/>
<xsd:element name="ref-set" type="Tref-collection"/>
```

5.7 Extender Capability

The Core OSGi specification defines a capability namespace for extender implementations. To enable Blueprint extendees to express a requirement for a Blueprint extender and also ensure classpath consistency with the chosen extender, the Blueprint extender must now specify the following capability:

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

The Blueprint container must not extend a bundle that is wired to another provider of the Blueprint extender capability.

A Require-Capability header that wires to this extender capability opts the bundle in to being processed by the blueprint extender. An example of the Require-Capability headers is as follows:

```
Require-Capability: osgi.extender; filter:="(osgi.extender=osgi.blueprint)"; ;
 path:List<String>=”lib/account.xml, security.bp, cnf/*.xml”
```

The path attribute follows the same pattern as the Bundle-Blueprint header described in section 121.3.4. Unlike, Bundle-Blueprint, absence of a path attribute means the blueprint extender searches for the blueprint xmls in the default location (i.e. OSGI-INF/blueprint/*.xml.

5.8 Allow <property> elements in Service Managers

The current mechanism for defining service properties in Service Managers using the <service-properties> elements and <entry> sub-elements is quite cumbersome, and is designed with the idea that the properties will be put into a Map.

Service Managers should allow one or more <property> elements to be defined, in the same way that Bean Managers do. All defined properties will be used as the service properties when the service is registered.

If a service manager is configured both with the existing <service-properties> mechanism, and also with the <property> elements, these will be merged into a single Map.

If there are any duplicate keys, <property> elements take precedence over <service-properties>.

The Property element will also have the valueType attribute as part of requirement Blueprint13

5.9 Blueprint Extender Configuration (Bug 2484)

Best practice design for extenders is for them to require an opt-in header specified in the extendee bundle's manifest. Blueprint 1.0, however, treats the Bundle-Blueprint header as an opt-out (when no value is specified). This means the default behavior for Blueprint 1.0 is to search all bundles for blueprint configuration, unless they contain the opt-out header, resulting in significant performance issues for large deployments. Rather than change the default blueprint extender behavior this specification adds the ability to configure the extender behavior.

Blueprint 1.0 also has a number of other configurations provided on the Bundle-Blueprint header or in the Blueprint XMLs. These are arguably better suited to being configured on the extender, rather than per-Blueprint. For example, blueprint.timeout, (grace perioid timeout), default-timeout, default-activation, default-availability.

The Blueprint extender is configured through configuration admin. The pid for the Blueprint extender configuration dictionary is `osgi.blueprint.Extender`.

5.9.1 Extendee Header Behavior

The extendee header behavior can be configured by setting the `osgi.blueprint.header` in the `osgi.blueprint.Extender` configuration dictionary. The default value is `OptOut` and configures the extender to behave as defined by the Blueprint 1.0 specification.

The configuration value of `OptIn` switches the default and opt-out behaviors of the extender, specifically:

- Absence of the `Bundle-Blueprint` header means a bundle is not processed for Blueprint configurations.
- If the `Bundle-Blueprint` header is specified with no value then the extender must search for Bluperint configurations in the default location.

The remaining extender processing of the `Bundle-Blueprint` header is unchanged.

5.9.2 ~~TODO: Other things configured through the dictionary~~

5.10 Add “Default” implementations for blueprint reference managers

Blueprint reference managers are responsible for locating and tracking suitable services from the OSGi service registry. They are also required to produce a proxy object which wraps the tracked service, as and when the tracked service becomes unavailable a new service replaces it inside the proxy. No reinjection of the proxy object is required.

If the tracked service becomes unavailable and there is no suitable replacement then invocations of the proxy object are required to wait until a replacement is found, or throw ServiceUnavailableException if this wait period times out.

This functionality could be usefully extended with the concept of “default service implementations”. Default service implementations can be thought of as locally visible services with the minimum possible ranking. This means that they aren't visible to other components, will never take precedence over a real service, can be used whenever no suitable service is available.

5.10.1 Declaring a default implementation

A default service implementation can be declared for a reference manager either by using an attribute to refer to an existing blueprint component, or by declaring an inline bean inside a <default> element.

Example 1:

```
<reference interface="java.util.List">
  <default>
    <bean class="java.util.ArrayList"/>
  <default>

</reference>
```

Example 2:

```
<bean id="example" class="java.util.ArrayList"/>

<reference default="example" interface="java.util.List"/>
```

When a default service implementation is declared in this way it is important that the default object be of the correct type to match the interfaces listed in the reference manager. If this is not the case then the blueprint container implementation is required to throw a `ComponentDefinitionException` when creating the reference manager.

5.10.2 Default implementations and the Null Proxy pattern

For some services it may be difficult to provide a suitable default implementation. Equally blueprint managed bundles may not wish to make themselves providers of the service that they are using (for example the HttpService), because it will significantly restrict the range of implementations with which they are compatible.

In this case the Null Proxy pattern can be used to support clients that want the benefits of a default implementation, but without the effort of implementing a default. The null proxy pattern involves generating a proxy object that performs no action, and returns `null` or null-like values, for all method calls. This would mean that all void methods do nothing, all methods that return numeric primitives return zero, all boolean methods return false, and all methods that return references return `null`.

The null proxy pattern can be enabled in blueprint using a new Environment Manager “blueprintNullProxy”, which has prototype scope. Whenever this environment manager is used the blueprint container must create a new null proxy matching the expected type of the receiver. For example in the following:

```
<reference default="blueprintNullProxy" interface="java.util.List"/>
```

The type of the blueprintNullProxy created would be `java.util.List`.

5.11 Support of new OSGi Service Scopes

RFC0195 introduces the concept of service scopes, and introduces PrototypeServiceFactory, which is a new subtype of ServiceFactory~~, PrototypeServiceFactory~~, that can return multiple instances of a particular service o~~O~~bject to the same consuming bundle depending on the scopes of the service o~~O~~bject~~, the service~~ and the reference that is consuming the service(s).

The scope of the service will be determined by the scope attribute on the bean, which can have a value of Singleton, Bundle or Prototype. A Scope of Singleton means that the Service Manager will return the same instance of the bean for all references in the Blueprint Container, a bundle scope will return the same instance of the bean for all references in a particular bundle, and a different instance for each bundle, and Prototype scope will return a different instance to all references in all bundles.

The default scope for a service is singleton, as this is the default scope for a bean.

The destroy_method on a bean manager will now get called whenever the unget method of the underlying ServiceFactory, that is created by the ServiceManager, is called. Singleton services will only get destroyed when the Blueprint container shuts down.

Note: It was discussed on a previous call about having a new unget-method attribute, or whether we should just use the existing destroy-method attribute on the Bean manager, for removal of the bean in non-singleton case

Services returned from the new PrototypeServiceFactory will have the same lifecycle as any other service, so the Container must track the consumed services and release them when the bundle is stopped.The Blueprint Container must also ensure that all services are registered with a service.scope service-property that indicates the scope of the service.- This will be determined by the scope of the bean.

We will also introduce a scope attribute on the reference manager that will be used to indicate the scope of services that can be selected. The scope values are singleton, bundle and prototype. If no scope attribute is set, the reference will select services of any scope.

References should not be injected with ServiceObject services.

The scopes of services that the reference manager can reference can also be defined via the filter attribute, using the service.scope service property.

5.12 Allow types to be defined in service-properties and Maps

If you want to define a map entry or a service property whose value isn't a String, then you need to use a nested <value> element in order to be able to set the type, e.g.

```
<entry key=”3”>
   <value type="org.osgi.framework.Version">3.14</value>
</entry>
```

To simplify this process a valueType attribute will be available on the <entry> element so that the above example can be simplified to:

<entry key=”3” value=”3.14” valueType="org.osgi.framework.Version"/>

Map values can be of any type, but service property values may only be one of the following types:

- *Primitives Number* – int, long, float, double, byte, short, char, boolean
- *Scalar* – String, Integer, Long, Float, Double, Byte, Short, Character, Boolean.
- *Array* – An array of either the allowable primitive or scalar types.
- *Collection* – An object implementing the Collection interface that contains scalar types.

The `TservicePropertyEntry` and `TmapEntry` definitions in the blueprint schema will need to be amended for this new attribute.

The valueType attribute will also be defined on the <property> element that requirement Blueprint8 introduces, where service-properties can be defined with property elements.

5.13 Blueprint 1.1 Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--
```

```
    /*
    * $Id: de3c851a280f003f52ccb3d325806f3f084b503f $
    *
    * Copyright (c) OSGi Alliance (2008, 2009). All Rights Reserved.
    *
    * Licensed under the Apache License, Version 2.0 (the "License");
    * you may not use this file except in compliance with the License.
    * You may obtain a copy of the License at
    *
    *      http://www.apache.org/licenses/LICENSE-2.0
    *
    * Unless required by applicable law or agreed to in writing, software
    * distributed under the License is distributed on an "AS IS" BASIS,
    * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    * See the License for the specific language governing permissions and
    * limitations under the License.
    */
    -->
<!-- xxx changed schema version xxx -->
<xsd:schema xmlns="http://www.osgi.org/xmlns/blueprint/v1.1.0"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.osgi.org/xmlns/blueprint/v1.1.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="1.1.0">

    <xsd:annotation>
        <xsd:documentation>
            <![CDATA[
                This is the XML Schema for the OSGi Blueprint service 1.0.0
                development descriptor.  Blueprint configuration files
                using this schema must indicate the schema using the
                blueprint/v1.0.0 namespace.  For example,

                <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

                if used as a qualified namespace, "bp" is the recommended
                namespace prefix.
            ]]>
        </xsd:documentation>
    </xsd:annotation>

    <!-- Schema elements for core component declarations -->

    <xsd:complexType name="Tcomponent" abstract="true">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                    The Tcomponent type is the base type for top-level
                    Blueprint components.  The <bean> <reference>, <service>,
                    and <reference-list> elements are all derived from
                    the Tcomponent type.  This type defines an id attribute
                    that is used create references between different components.
                    Component elements can also be inlined within other component
                    definitions.  The id attribute is not valid when inlined.
```

```
                ]]>
            </xsd:documentation>
        </xsd:annotation>

        <xsd:attribute name="id" type="xsd:ID" />

        <xsd:attribute name="activation" type="Tactivation">
            <xsd:annotation>
                <xsd:documentation>
                    <![CDATA[
                    The activation attribute for this component.  This can either
                    be "eager" or "lazy".  If not specified, it
                    defaults to default-activation attribute of the enclosing
                    <blueprint> element.
                    ]]>
                </xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>

        <xsd:attribute name="depends-on" type="TdependsOn">
            <xsd:annotation>
                <xsd:documentation>
                    <![CDATA[
                    depends-on identifies (by id) other components that this
component
                    depends on.  The component only be activated after the
                    depends-on components are successfully activated.  Also, if
there
                    are <reference> or <reference-list> elements with unstatisfied
                    manadatory references, then the depends-on relationship will
also
                    be used to determine whether this service is enabled or not.
                    ]]>
                </xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>

    <xsd:element name="blueprint" type="Tblueprint">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The <blueprint> element is the root element for a blueprint
                configuration file.  A blueprint configuration has two sections.
                The first section (contained within the <type-converters> element)
                identifies components that are used for converting values into
                different target types.  The type converters are optional, so
                the file does not need to specify a type converter section.

                Following the type converters are the component definitions.
                Components are <bean>, <service>, <reference>, and
                <reference-list> elements that identify the bundle components that
will
                be managed by the blueprint service.
```

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

<!-- xxx added top level service element xxx -->
    <xsd:element name="service" type="Tservice">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                This <service> element is to allow custom namespaces to
                inline this element in their own schema definitions.

                This element is not intended for direct use by blueprint bundles.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

<!-- xxx added top level reference-list element xxx -->
    <xsd:element name="reference-list" type="Treference-list">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                This <reference-list> element is to allow custom namespaces to
                inline this element in their own schema definitions.

                This element is not intended for direct use by blueprint bundles.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

<!-- xxx added top level bean element xxx -->
    <xsd:element name="bean" type="Tbean">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                This <bean> element is to allow custom namespaces to
                inline this element in their own schema definitions.

                This element is not intended for direct use by blueprint bundles.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

 <!-- xxx added top level reference element xxx -->
    <xsd:element name="reference" type="Treference">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                This <reference> element is to allow custom namespaces to
                inline this element in their own schema definitions.
```

```
                This element is not intended for direct use by blueprint bundles.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:complexType name="Tblueprint">
        <xsd:sequence>
            <xsd:element name="description" type="Tdescription"
                minOccurs="0" />
            <xsd:element name="type-converters" type="Ttype-converters"
                minOccurs="0" maxOccurs="1" />
            <!-- top-level components -->
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="service" type="Tservice" />
                <xsd:element name="reference-list" type="Treference-list" />
                <xsd:element name="bean" type="Tbean" />
                <xsd:element name="reference" type="Treference" />
                <xsd:any namespace="##other" processContents="strict" />
            </xsd:choice>
        </xsd:sequence>

        <!-- Defaults-->
        <xsd:attribute name="default-activation" default="eager"
            type="Tactivation">
            <xsd:annotation>
                <xsd:documentation>
                    <![CDATA[
                    Specifies the default activation setting that will be defined
                    for components.  If not specified, the global default is
"eager".
                    Individual components may override the default value.
                    ]]>
                </xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="default-timeout" type="Ttimeout"
            default="300000">
            <xsd:annotation>
                <xsd:documentation>
                    <![CDATA[
                    Specifies the default timeout value to be used when operations
                    are invoked on unstatisfied service references.  If the
                    reference does not change to a satisfied state within the
timeout
                    window, an error is raised on the method invocation.  The
                    default timeout value is 300000 milliseconds and individual
                    <reference> element can override the specified configuration
                    default.  A value of 0 means wait indefinitely, and a value of
-1
                    means do not wait.
                    ]]>
<!-- xxx updated the description above by adding last sentence xxx -->
```

```
                </xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="default-availability" type="Tavailability"
            default="mandatory">
            <xsd:annotation>
                <xsd:documentation>
                    <![CDATA[
                    Specifies the default availability value to be used for
                    <reference>, and <reference-list> components.  The
                    normal default is "mandatory", and can be changed by individual
                    service reference components.
                    ]]>
                </xsd:documentation>

            </xsd:annotation>
        </xsd:attribute>
        <xsd:anyAttribute namespace="##other"
            processContents="strict" />
    </xsd:complexType>

    <xsd:complexType name="Ttype-converters">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The type used for the <type-converters> element.  The
                <type-converters> section is a set of <bean>, <ref>, or
                <reference> elements that identify the type converter components.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="bean" type="Tbean" />
            <xsd:element name="reference" type="Treference" />
            <xsd:element name="ref" type="Tref" />
            <xsd:any namespace="##other" processContents="strict" />
        </xsd:choice>
    </xsd:complexType>

    <!--
        Components that provide a reasonable target for injection used for
        listeners, etc.
    -->

    <xsd:group name="GtargetComponent">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                A target component is one that can be a target for a
                listener, registration-listener or service elements.
                This is used in contexts where the requirement is a single
                provided object that will implement a particular interface.
                The provided object is obtained either from a <ref> element
```

```
                or an inlined <bean> or <reference>.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice>
            <xsd:element name="bean" type="Tinlined-bean" />
            <xsd:element name="reference" type="Tinlined-reference" />
            <xsd:element name="ref" type="Tref" />
            <xsd:any namespace="##other" processContents="strict" />
        </xsd:choice>
    </xsd:group>

    <xsd:group name="GallComponents">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                An all components is used in contexts where all component element
                types are values.  The set of component elements contains
                <bean>, <service>, <reference>, <reference-list> and <ref>.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice>
            <xsd:element name="service" type="Tinlined-service" />
            <xsd:element name="reference-list" type="Tinlined-reference-list" />
            <xsd:group ref="GtargetComponent" />
        </xsd:choice>
    </xsd:group>

    <xsd:group name="GbeanElements">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                A bean elements is a reusable definition of the elements allowed on
                a <bean> element.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="description" type="Tdescription"
                minOccurs="0" />
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="argument" type="Targument" />
                <xsd:element name="property" type="Tproperty" />
                <xsd:any namespace="##other" processContents="strict" />
            </xsd:choice>
        </xsd:sequence>
    </xsd:group>

    <xsd:complexType name="Tbean">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The type definition for a <bean> component.  The <bean>
```

```
                attributes provide the characteristics for how to create a
                bean instance.  Constructor arguments and injected properties
                are specified via child <argument> and <property> elements.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="Tcomponent">
                <xsd:group ref="GbeanElements" />
                <xsd:attribute name="class" type="Tclass" />
                <xsd:attribute name="init-method" type="Tmethod" />
                <xsd:attribute name="destroy-method" type="Tmethod" />
                <xsd:attribute name="factory-method" type="Tmethod" />
                <xsd:attribute name="factory-ref" type="Tidref" />
                <xsd:attribute name="scope" type="Tscope" />
                <xsd:anyAttribute namespace="##other"
                    processContents="strict" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="Tinlined-bean">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The Tinlined-bean type is used for inlined (i.e. non top level)
                <bean> elements.  Those elements have some restrictions on
                the attributes that can be used to define them.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:restriction base="Tbean">
                <xsd:group ref="GbeanElements" />
                <xsd:attribute name="id" use="prohibited" />
                <xsd:attribute name="depends-on" type="TdependsOn" />
                <xsd:attribute name="activation" use="prohibited"
                    fixed="lazy" />
                <xsd:attribute name="class" type="Tclass" />
                <xsd:attribute name="init-method" type="Tmethod" />
                <xsd:attribute name="destroy-method" use="prohibited" />
                <xsd:attribute name="factory-method" type="Tmethod" />
                <xsd:attribute name="factory-ref" type="Tidref" />
                <xsd:attribute name="scope" use="prohibited" />
                <xsd:anyAttribute namespace="##other"
                    processContents="strict" />
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="Targument">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
```

```
                An argument used to create an object defined by a <bean>
                component.  The <argument> elements are the arguments for the
                bean class constructor or passed to the bean factory method.

                The type, if specified, is used to disambiguate the constructor
                or method signature.  Arguments may also be matched up with
                arguments by explicitly specifying the index position.  If the
                index is used, then all <argument> elements for the bean must
                also specify the index.

                The value and ref attributes are convenience shortcuts to make
                the <argument> tag easier to code.  A fuller set of injected
                values and types can be specified using one of the "value"
                type elements.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="description" type="Tdescription"
                minOccurs="0" />
            <xsd:group ref="Gvalue" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="index" type="xsd:nonNegativeInteger" />
        <xsd:attribute name="type" type="Ttype" />
        <xsd:attribute name="ref" type="Tidref" />
        <xsd:attribute name="value" type="TstringValue" />
    </xsd:complexType>

    <xsd:complexType name="Tproperty">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                A property that will be injected into a created <bean>
                component.  The <property> elements correspond to named
                JavaBean setting methods for a created bean object.

                The value and ref attributes are convenience shortcuts to make
                the <argument> tag easier to code.  A fuller set of injected
                values and types can be specified using one of the "value"
                type elements.

                The valueType attribute allows the type of the attribute to be
                specified.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="description" type="Tdescription"
                minOccurs="0" />
            <xsd:group ref="Gvalue" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="name" type="Tmethod" use="required" />
        <xsd:attribute name="ref" type="Tidref" />
        <xsd:attribute name="value" type="TstringValue" />
```

```
<!-- xxx added valueType xxx -->
        <xsd:attribute name="valueType" type="Ttype" />
    </xsd:complexType>

    <xsd:complexType name="Tkey">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The Tkey type defines the element types that are permitted
                for Map key situations.  These can be any of the "value"
                types other than the <null> element.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:group ref="GnonNullValue" />
    </xsd:complexType>

    <!-- reference -->
    <xsd:complexType name="Treference">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The Treference type defines the <reference> element.  These
                are instances of the TserviceReference type, with the addition
                of a timeout attribute.  If the timeout is not specified,
                the default-timeout value is inherited from the encapsulating
                <blueprint> definition.  A timeout value of 0 means wait
                indefinitely, and a value of -1 means do not wait.
                ]]>
<!-- xxx updated
 description above by adding last sentence xxx -->
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="TserviceReference">
                <xsd:sequence>
<!-- xxx added inlined default element -->
                    <xsd:element name="default" type="Tinlined-default-bean"
minOccurs="0" />
                    <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
                        processContents="strict" />
                </xsd:sequence>
                <xsd:attribute name="timeout" type="Ttimeout" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

<!-- xxx added this type for inlined default bean on references -->
    <xsd:complexType name="Tinlined-default-bean">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The Tinlined-default-bean type is used to inline a default bean
implementation
```

```
                for a <reference>.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
                <xsd:element name="bean" type="Tinlined-bean" />
                <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
                processContents="strict" />
        </xsd:sequence>
        <xsd:anyAttribute namespace="##other"
            processContents="strict" />
    </xsd:complexType>

    <xsd:complexType name="Tinlined-reference">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The Tinlined-reference type is used for inlined (i.e. non top level)
                <reference> elements.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:restriction base="Treference">
                <xsd:sequence>
                    <xsd:group ref="GserviceReferenceElements" />
                    <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
                        processContents="strict" />
                </xsd:sequence>
                <xsd:attribute name="id" use="prohibited" />
                <xsd:attribute name="depends-on" type="TdependsOn" />
                <xsd:attribute name="activation" use="prohibited"
                    fixed="lazy" />
                <xsd:attribute name="interface" type="Tclass" />
                <xsd:attribute name="filter" type="xsd:normalizedString" />
                <xsd:attribute name="component-name" type="Tidref" />
                <xsd:attribute name="availability" type="Tavailability" />
                <xsd:attribute name="timeout" type="Ttimeout" />
                <xsd:anyAttribute namespace="##other"
                    processContents="strict" />
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- reference-list -->
    <xsd:complexType name="Treference-list">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The Treference-list builds in the characteristics of the
                TserviceReference type to define characteristics of the
                <reference-list>.  This adds in the characteristics that
                only apply to collections of references (e.g., member-type).
                ]]>
```

```
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="TserviceReference">
                <xsd:sequence>
                    <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
                        processContents="strict" />
                </xsd:sequence>
                <xsd:attribute name="member-type" type="Tservice-use"
                    default="service-object">
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="Tinlined-reference-list">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The Tinlined-reference-list type is used for inlined (i.e. non top
level)
                <reference-list> elements.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:restriction base="Treference-list">
                <xsd:sequence>
                    <xsd:group ref="GserviceReferenceElements" />
                    <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
                        processContents="strict" />
                </xsd:sequence>
                <xsd:attribute name="id" use="prohibited" />
                <xsd:attribute name="depends-on" type="TdependsOn" />
                <xsd:attribute name="activation" use="prohibited"
                    fixed="lazy" />
                <xsd:attribute name="interface" type="Tclass" />
                <xsd:attribute name="filter" type="xsd:normalizedString" />
                <xsd:attribute name="component-name" type="Tidref" />
                <xsd:attribute name="availability" type="Tavailability" />
                <xsd:attribute name="member-type" type="Tservice-use"
                    default="service-object" />
                <xsd:anyAttribute namespace="##other"
                    processContents="strict" />
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- Reference base class -->
    <xsd:complexType name="TserviceReference">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TserviceReference is the base element type used for <reference>
```

```
                and <reference-list> elements.  This type defines all of the
                characteristics common to both sorts of references.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="Tcomponent">
                <xsd:sequence>
                    <xsd:group ref="GserviceReferenceElements" />
                </xsd:sequence>

                <xsd:attribute name="interface" type="Tclass">
                    <xsd:annotation>
                        <xsd:documentation>
                            <![CDATA[
                            The interface that the OSGi service must implement and
that will be
                            implemented by the proxy object.
                            This attribute is optional.
                            ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="filter" type="xsd:normalizedString">
                    <xsd:annotation>
                        <xsd:documentation>
                            <![CDATA[
                            A filter string used to narrow the search for a
matching service
                            reference.
                            ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="component-name" type="Tidref">
                    <xsd:annotation>
                        <xsd:documentation>
                            <![CDATA[
                            An optional specifier that can be used to match a
service definition
                            to one created by a specific blueprint component.
                            ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="availability" type="Tavailability">
                    <xsd:annotation>
                        <xsd:documentation>
                            <![CDATA[
                            Use to control the initial processing of service
references at
                            blueprint context startup.  "mandatory" indicates the
context
                            should not start unless the service is available within
```

```
the
                                specified context startup period.  "optional" indicates
availability
                                of this service is not a requirement at bundle startup.

                                NOTE:  No default is specified because this can be
overridden
                                by the default-availability attribute of the
<blueprint> element.
                                ]]>
                            </xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
<!-- xxx added scope attribute xxx -->
                    <xsd:attribute name="scope" type="Tscope">
                        <xsd:annotation>
                            <xsd:documentation>
                                <![CDATA[
                                An options specifier that can be used to match to a
service of a specific
                                scope.  When not specified the dependency can be
matched to services of
                                any scope.
                                ]]>
                            </xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
<!-- xxx added default attribute xxx -->
                    <xsd:attribute name="default" type="Tidref">
                        <xsd:annotation>
                            <xsd:documentation>
                                <![CDATA[
                                A default bean implementation to use if the service
reference cannot be
                                bound to a service.
                                ]]>
                            </xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:anyAttribute namespace="##other"
                        processContents="strict" />
                </xsd:extension>
            </xsd:complexContent>

        </xsd:complexType>

        <xsd:group name="GserviceReferenceElements">
            <xsd:sequence>
                <xsd:element name="description" type="Tdescription"
                    minOccurs="0" />
                <!-- listener -->
                <xsd:element name="reference-listener" type="TreferenceListener"
                    minOccurs="0" maxOccurs="unbounded">
                    <xsd:annotation>
```

```
                    <xsd:documentation>
                        <![CDATA[
                        A definition of a listener that will watch for bind/unbind
events
                        associated with the service reference.  The targetted
listener can
                        be a <ref> to a <bean> or <reference> element, or an inline
                        <bean> or <reference>.
                        ]]>
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
    </xsd:group>

    <xsd:complexType name="TreferenceListener">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TReferenceListener defines a reference listener that is attached
                to a <reference> or <reference-list> element.  The listener
                object can be specified as a <ref> or as an inline <bean> or
                <reference> component.  Listener events are mapped to the indicated
                bind or unbind methods.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:group ref="GtargetComponent" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="ref" type="Tidref" />
        <xsd:attribute name="bind-method" type="Tmethod" />
        <xsd:attribute name="unbind-method" type="Tmethod" />
    </xsd:complexType>

    <xsd:simpleType name="Tactivation">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tactivation defines the activation type for components.  This is
used in this
                schema by the <blueprint> default-activation attribute and the
                activation attribute.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="eager" />
            <xsd:enumeration value="lazy" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="Tavailability">
        <xsd:annotation>
```

```
            <xsd:documentation>
                <![CDATA[
                Tavailability defines an availability attribute type.  This is used
in this
                schema by the <blueprint> default-availability attribute and the
                <reference> and <reference-list> availability attribute.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="mandatory" />
            <xsd:enumeration value="optional" />
        </xsd:restriction>
    </xsd:simpleType>

    <!-- service -->

    <xsd:complexType name="Tservice">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tservice is the type for services exported by this blueprint bundle.
                Services are sourced by either a <ref> to a <bean> component or an
                <inline> bean component.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="Tcomponent">
                <xsd:sequence>
                    <xsd:group ref="GserviceElements" />
                </xsd:sequence>
                <xsd:attribute name="interface" type="Tclass">
                    <xsd:annotation>
                        <xsd:documentation>
                            <![CDATA[
                            The interface that this OSGi service will provide.
                            ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="ref" type="Tidref">
                    <xsd:annotation>
                        <xsd:documentation>
                            <![CDATA[
                            The ref attribute can be used to specify the component
that provides
                            the object exported as an OSGi service.
                            ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="auto-export" type="TautoExportModes"
                    default="disabled">
```

```
                    <xsd:annotation>
                        <xsd:documentation>
                            <![CDATA[
                            If set to a value different from "disabled", the
Blueprint Container
                            will introspect the target to discover the set of
interfaces or classes
                            that the service will be registered under.
                            ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="ranking" type="xsd:int" default="0">
                    <xsd:annotation>
                        <xsd:documentation>
                            <![CDATA[
                            A service ranking value that is added to the service
properties
                            the service will be published with.
                            ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:anyAttribute namespace="##other"
                    processContents="strict" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="Tinlined-service">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The Tinlined-service type is used for inlined (i.e. non top level)
                <service> elements.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:restriction base="Tservice">
                <xsd:sequence>
                    <xsd:group ref="GserviceElements" />
                </xsd:sequence>
                <xsd:attribute name="id" use="prohibited" />
                <xsd:attribute name="depends-on" type="TdependsOn" />
                <xsd:attribute name="activation" use="prohibited"
                    fixed="lazy" />
                <xsd:attribute name="interface" type="Tclass" />
                <xsd:attribute name="ref" type="Tidref" />
                <xsd:attribute name="auto-export" type="TautoExportModes"
                    default="disabled" />
                <xsd:attribute name="ranking" type="xsd:int" default="0" />
                <xsd:anyAttribute namespace="##other"
                    processContents="strict" />
```

```
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:group name="GbaseServiceElements">
        <xsd:sequence>
            <xsd:element name="description" type="Tdescription"
                minOccurs="0" />
            <xsd:element name="interfaces" type="Tinterfaces"
                minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>
                        <![CDATA[
                        A collection of one or more interface class names this
service
                        will be registered under.  The <service> element also has
                        a shortcut interface attribute for the usual case of just
                        a single interface being used.  This also cannot be used if
                        the auto-export attribute is used.
                        ]]>
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>

            <xsd:element name="service-properties" type="TserviceProperties"
                minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>
                        <![CDATA[
                        The service provided when the service is registered.  The
service
                        properties are similar to map elements, but the keys must
always
                        be strings, and the values are required to be in a narrower
range.
                        ]]>
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>

<!-- xxx added <property> -->
            <xsd:element name="property" type="TserviceProperty"
                minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>
                        <![CDATA[
                        The <property> element used to specify service properties.
This
                        can be used in conjunection with <service-properties>.
                        ]]>
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="registration-listener" type="TregistrationListener"
```

```
                minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>
                        <![CDATA[
                        A set of 0 or more registration listeners attached to this
service
                        component.  The registration listeners will be notified
whenever the
                        service is registered or unregistered from the framework
service
                        registry.
                        ]]>
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>

        </xsd:sequence>
    </xsd:group>

    <xsd:group name="GserviceElements">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                A set of service elements.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:group ref="GbaseServiceElements" />
            <xsd:group ref="GtargetComponent" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>
                        <![CDATA[
                        A service definition can use any of the target types as an
inline element
                        as well.
                        ]]>
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:group>
        </xsd:sequence>
    </xsd:group>

    <xsd:complexType name="TregistrationListener">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                A registration listener definition.  The target registration
listener
                can be either a <ref> to a <bean> or <service> component, or an
inline
                <bean> or <service> component definition.  The registration-method
and
                unregistration-method attributes define the methods that will be
```

```
called
                for the respective events.

                For the very common case of using a <ref> to a listener component,
the
                ref attribute may also be used as a shortcut.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:group ref="GtargetComponent" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="ref" type="Tidref" />
        <xsd:attribute name="registration-method" type="Tmethod" />
        <xsd:attribute name="unregistration-method" type="Tmethod" />
    </xsd:complexType>

    <!-- Values -->

    <xsd:group name="Gvalue">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The set of "value" types that can be used in any place a value
                can be specified.  This set includes the <ref> and <idref>
elements, any of the
                component types (<bean>, <service>, etc.) as inline components, the
                generic <value> element for types sourced from string values, any
of the
                collection types (<set>, <list>, <array>, <map>, <props>), and the
                <null> type to inject a null value.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice>
            <xsd:group ref="GnonNullValue" />
            <xsd:element name="null" type="Tnull" />
        </xsd:choice>
    </xsd:group>

    <xsd:complexType name="Tnull">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The definition for a <null> value type.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
    </xsd:complexType>

    <xsd:group name="GnonNullValue">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
```

```
                The set of "value" types that can be used in any place a non-null
value
                can be specified.  This set includes the <ref> and <idref>
elements, any of the
                component types (<bean>, <service>, etc.) as inline components, the
                generic <value> element for types sourced from string values, and
any of the
                collection types (<set>, <list>, <array>, <map>, <props>).

                The <null> type is NOT a member of this group.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice>
            <xsd:group ref="GallComponents" />
            <xsd:element name="idref" type="Tref" />
            <xsd:element name="value" type="Tvalue" />
            <xsd:element name="list" type="Tcollection" />
            <xsd:element name="set" type="Tcollection" />
            <xsd:element name="map" type="Tmap" />
            <xsd:element name="array" type="Tcollection" />
            <xsd:element name="props" type="Tprops" />
        </xsd:choice>
    </xsd:group>

    <xsd:complexType name="Tref">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tref is the type used for <ref> elements.  This specifies a required
                component id for the reference component.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="component-id" type="Tidref" use="required" />
    </xsd:complexType>

    <xsd:complexType name="Tvalue" mixed="true">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tvalue is the type used for <value> elements.  The <value> element
                is used for types that can be created from a single string value.
                The string value is the data value for the element.  The optional
                type attribute allows a target conversion value to be explicitly
                specified.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="type" type="Ttype" />
    </xsd:complexType>

    <!-- Collection Values -->
```

```
    <xsd:complexType name="TtypedCollection">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TtypeCollection defines comment attributes shared among different
                collection types that allow a default value type to be specified.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="value-type" type="Ttype" />
    </xsd:complexType>

    <xsd:complexType name="Tcollection">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tcollection is the base schema type for different ordered collection
                types.  This is shared between the <array>, <list>, and <set>
elements.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="TtypedCollection">
                <xsd:group ref="Gvalue" minOccurs="0" maxOccurs="unbounded" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="Tprops">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tprops is the type used by the <props> value element.  The prop
elements
                are pairs of string-valued keys and values.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="prop" type="Tprop" minOccurs="0"
                maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Tprop" mixed="true">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tprop is a single property element for a <props> value type.  The
property
                value can be specified using either the attribute, or as value data
for
                the property element.
```

```
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="key" type="TstringValue" use="required" />
        <xsd:attribute name="value" type="TstringValue" />
    </xsd:complexType>

    <!-- 'map' element type -->
    <xsd:complexType name="Tmap">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tmap is the base type used for <map> elements.  A map may have a
                default value type specified, so it inherits from the
TtypeCollection
                type.  A key type can also be specified, and the map members are
                created from the entry elements, which require a key/value pair.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="TtypedCollection">
                <xsd:sequence>
                    <xsd:element name="entry" type="TmapEntry" minOccurs="0"
                        maxOccurs="unbounded" />
                </xsd:sequence>
                <xsd:attribute name="key-type" type="Ttype" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <!-- 'entry' element type -->
    <xsd:complexType name="TmapEntry">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TmapEntry is used for <entry> elements nested inside of a <map>
element.
                Each <entry> instance defines a key/value pair that will be added
to the
                Map.  Both the keys and values may be arbitrary types.  Keys must
not
                be <null> but <null> is permitted for entry values.  A default type
                can be specified for both the keys and the values, but individual
keys
                or values can override the default.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="key" type="Tkey" minOccurs="0" />
            <xsd:group ref="Gvalue" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="key" type="TstringValue" />
```

```
        <xsd:attribute name="key-ref" type="Tidref" />
        <xsd:attribute name="value" type="TstringValue" />
        <xsd:attribute name="value-ref" type="Tidref" />
<!-- xxx added valueType attribute -->
        <xsd:attribute name="valueType" type="Ttype" />
    </xsd:complexType>

    <!-- 'service property' element type -->
    <xsd:complexType name="TserviceProperties">
        <xsd:annotation>
            <xsd:documentation>

                <![CDATA[
                TserviceProperties is used for <service-properties> elements.
                The syntax is similar to what is defined for <map>, but keys must be
                string values and there are no type defaults that can be specified.
                created from the entry elements, which require a key/value pair.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="entry" type="TservicePropertyEntry"
                minOccurs="0" maxOccurs="unbounded" />
            <xsd:any namespace="##other" processContents="strict"
                minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>

    <!-- 'entry' element type -->
    <xsd:complexType name="TservicePropertyEntry">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TservicePropertyEntry is an entry value used for the <service-
properties>
                element.  This does not allow a child <key> element and there are no
                key-ref or value-ref attributes.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:group ref="Gvalue" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="key" type="TstringValue" use="required" />
        <xsd:attribute name="value" type="TstringValue" />
<!-- xxx added valueType attribute -->
        <xsd:attribute name="valueType" type="Ttype" />
    </xsd:complexType>

<!-- xxx added TserviceProperty type -->
    <!-- service 'property' element type -->
    <xsd:complexType name="TserviceProperty">
        <xsd:annotation>
            <xsd:documentation>
```

```
                <![CDATA[
                TserviceProperty is used to specify a <property> on a <service>
                element.  This does not allow a child <key> element and there are no
                key-ref or value-ref attributes.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:group ref="Gvalue" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="key" type="TstringValue" use="required" />
        <xsd:attribute name="value" type="TstringValue" />
        <xsd:attribute name="valueType" type="Ttype" />
    </xsd:complexType>

    <!-- General types -->

    <xsd:complexType name="Tdescription" mixed="true">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                A generic <description> element type to allow documentation to
added to the
                blueprint configuration.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="0" maxOccurs="unbounded" />
    </xsd:complexType>

    <xsd:complexType name="Tinterfaces">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                The type definition for the <interfaces> element used for <service>
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="1" maxOccurs="unbounded">
            <xsd:element name="value" type="TinterfaceValue" />
        </xsd:choice>
    </xsd:complexType>

    <xsd:simpleType name="TinterfaceValue">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TinterfaceValue is used for subelements of the <interfaces> element.
                This is just a <value>xxxxx</value> element where the contained
                value is the name of an interface class.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="Tclass" />
```

```
    </xsd:simpleType>

    <xsd:simpleType name="Tclass">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tclass is a base type that should be used for all attributes that
                refer to java class names.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:NCName" />
    </xsd:simpleType>

    <xsd:simpleType name="Ttype">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Ttype is a base type that refer to java types such as classes or
                arrays.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:token">
            <xsd:pattern value="[\i-[:]][\c-[:]]*(\[\])*" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="Tmethod">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tmethod is a base type that should be used for all attributes that
                refer to java method names.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:NCName" />
    </xsd:simpleType>

    <!--
        Should be used for all attributes and elements that refer to method
        names
    -->
    <xsd:simpleType name="Tidref">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Tidref is a base type that should be used for all attributes that
                refer to component ids.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:NCName" />
```

```
    </xsd:simpleType>

    <xsd:simpleType name="TstringValue">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TstringValue is a base type that should be used for all attributes
that
                refer to raw string values
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:normalizedString" />
    </xsd:simpleType>

    <xsd:simpleType name="TautoExportModes">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TautoExportModes is a base type that should be used for export-mode
                attributes.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="disabled" />
            <xsd:enumeration value="interfaces" />
            <xsd:enumeration value="class-hierarchy" />
            <xsd:enumeration value="all-classes" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="Ttimeout">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Ttimeout is a base type that should be used for all attributes that
                specify timeout values
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:unsignedLong" />
    </xsd:simpleType>

    <xsd:simpleType name="TdependsOn">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                TdependsOn is a base type that should be used for all attributes
that
                specify depends-on relationships
                ]]>
            </xsd:documentation>
        </xsd:annotation>
```

```
        <xsd:restriction>
            <xsd:simpleType>
                <xsd:list itemType="Tidref" />
            </xsd:simpleType>
            <xsd:minLength value="1" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="Tscope">
        <xsd:union>
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="singleton" />
<!-- xxx added bundle scope xxx -->
                    <xsd:enumeration value="bundle" />
                    <xsd:enumeration value="prototype" />
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:restriction base="xsd:QName">
                    <xsd:pattern value=".+:.+" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>

    <xsd:simpleType name="Tservice-use">
        <xsd:annotation>
            <xsd:documentation>
                <![CDATA[
                Indicates the type of object that will be placed within the
                reference collection.  "service-object" indicates the
                collection contains blueprint proxies for imported services.
                "service-reference" indicates the collection contains
                ServiceReference objects matching the target service type.
                ]]>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="service-object" />
            <xsd:enumeration value="service-reference" />
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

6 Considered Alternatives

7 Security Considerations

8 Document Support

8.1 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]. Woolf, Bobby (1998). "Null Object". In Martin, Robert; Riehle, Dirk; Buschmann, Frank. *Pattern Languages of Program Design 3*. Addison-Wesley

8.2 Author's Address

Name	Tim Mitchell
Company	IBM
Address	
Voice	
e-mail	tim.mitchell@uk.ibm.com

Name	Graham Charters
Company	IBM
Address	
Voice	
e-mail	charters@uk.ibm.com

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

8.3 Acronyms and Abbreviations

8.4 End of Document