RFC 150 - Fine Grained APIs and Versioned PIDs in Config Admin

Draft

18 Pages

Text in Red is here to help you. Delete it when you have followed the instructions.
The <RFC Title> and <Company> can be set from the File>Properties:User Defined menu. To update it onscreen, press F9. To update all of the fields in the document Select All (CTRL-A), then hit F9. Set the release level by selecting one from: Draft, Final Draft, Release. The date is set automagically when the document is saved. ***Note that FINAL DRAFT and RELEASE documents should only be released as PDF, NOT AS ODF DOCUMENTS***

Abstract

In today's OSGi Compendium specification, the API available to the Configuration Admin's ManagedService is coarse-grained and doesn't provide any delta information. Many of today's applications have a large set of configuration variables and would benefit from more fine-grained callbacks with additional metadata regarding what has changed.

Additionally, configuration PIDs are currently unversioned entities in the Configuration Admin Service, causing problems when multiple versions of the same bundle are installed simultaneously. This RFC contains the technical design for a solution to these problems and addresses the requirements described in RFP 119.

Copyright © Progress Software Corporation 2010
This contribution is made to the OSGi Alliance as MEMBER LICENSED MATERIALS pursuant to the terms of the OSGi Member Agreement and specifically the license rights and warranty disclaimers as set forth in Sections 3.2 and 12.1, respectively.
All company, brand and product names contained within this document may be trademarks that are the sole property of the respective owners.
The above notice must be included on all copies of this document that are made.

0 Document Information

0.1 Table of Contents

Copyright © Progress Software Corporation 2010 All Rights Reserved

0.2 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 8.1.

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

0.3 Revision History

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

Revision	Date	Comments
Initial	2 November 2009	David Bosschaert (Progress), David Grigglestone (Progress) initial version based on requirements from RFP 119.
0.1	December 2009	David Bosschaert (Progress) changed the design to incorporate feedback from the Portland F2F.
0.2	March 2010	David Bosschaert (Progress) incorporating feedback from Southampton F2F.
0.3	April 2010	David Bosschaert (Progress) feedback from Mountain View F2F.

1 Introduction

When using the Configuration Admin Service in larger enterprise applications certain shortcomings become apparent. Some of these shortcomings can be worked around, but doing so will cause duplicated code in bundles that use the Configuration Admin Service. Other shortcomings are more fundamental and prevent the Configuration Admin from being used in ways desirable for the enterprise application.

Copyright © Progress Software Corporation 2010 All Rights Reserved

This RFC contains the technical design for a solution to this problem and addresses the requirements described in RFP 119.

TODO: Configuration Plugins.

2 Application Domain

Configuration Management is a comprehensive part of many of today's enterprise applications. Often these applications use a sophisticated configuration management system which is used to enter, record and distribute configuration information.

The OSGi Configuration Admin specification is an appealing technology for such applications since its one of the few standards in the configuration domain.

This document describes proposed enhancements to the Configuration Admin specification to better align its capabilities with enterprise use-cases.

2.1 Terminology + Abbreviations

Configuration PID: The Configuration Admin Service requires that every configuration set (actually a map of name-value pairs) be assigned a unique, persistent, ID. Typically Service PIDs have been used for this purpose, but other PIDs are also possible. See also RFC 144.

Configuration Target: The target (bundle or service) that will receive the configuration information. For services, there are currently two types of targets: ManagedServiceFactory or ManagedService objects. The realization of the requirements in this document may introduce new configuration targets.

3 Problem Description

3.1 Provide Delta information in Configuration Target callbacks

When using the Configuration Admin Service to configure the components of large systems, it becomes apparent that certain APIs could be improved to better provide the consumers of configuration information with what has actually changed in the configuration.

Copyright © Progress Software Corporation 2010 All Rights Reserved

The current Configuration Admin specification always calls consumers of configuration information (implementors of the ManagedService or ManagedServiceFactory) with the full map of configuration information. Even thought only a subset of the configuration information may have changed. This requires every implementor of the ManagedService to keep a copy of the previous map and implement logic to compute the deltas between the current callback and the previous callback. It would benefit the system if this functionality was provided by the Configuration Admin Service.

3.2 Updates concerning multiple Configuration PIDs

Large systems can have hundreds or sometimes thousands of configuration variables. Typically such systems are modularized in some way (e.g. using OSGi bundles), where some configuration information is shared across modules and other configuration is private to the module.

The Configuration Admin Service allows a Managed Service to be a consumer of multiple configuration sets (configuration PIDs), but it does not provide an API capable of notifying the ManagedService of changes to multiple configuration sets in a single callback. This would be necessary if there is a dependency of configuration values across the configuration sets.

3.3 Multiple versions and multiple instances of Configuration data

In today's Configuration Admin specification, only one instance of Configuration Data exists in the system for a given PID. This can cause issues when multiple versions of the same bundle are installed in the system, especially if the configuration data required by these versions of the bundle is not compatible, or if the configuration values to be provided to the various instances of the bundle need to be different.

4 Requirements

FGCA00. The solution depends on a feature of the Configuration Admin Service introduced in RFC 144: the capability of multiple bundles being a able to consume a single Configuration PID.

FGCA01. The solution MUST provide delta information when calling back with configuration updates.

FGCA02. The solution MUST provide the configuration PID on all newly defined callbacks.

FGCA03. The solution MUST provide the capability of combining multiple configuration changes in a single callback to the ManagedService.

FGCA04. The solution MUST provide an API that allows a single update callback for multiple configuration PIDs on a particular Configuration Target.

FGCA05. The solution SHOULD be applied to both the factory configurations as well as normal configurations.

FGCA06. It MUST be possible to have multiple versions of configuration data exist concurrently for a given PID.

Copyright © Progress Software Corporation 2010 All Rights Reserved

FGCA07. It SHOULD be possible for a Configuration Target to specify a version range for a suitable configuration object, given a PID.

FGCA08. It MUST be possible for a Configuration Target to specify the exact version of a suitable configuration object, given a PID.

FGCA09. It MUST be possible for two versions of the same bundle to use separate configuration objects for a given PID if the configuration PID versions used are different.

FGCA10. It MUST be possible for two versions of the same bundle to share the same configuration object for a given PID if the configuration PID versions are the same.

FGCA11. The solution MUST be backwards compatible. This implies that the configuration consumer must also be able to use the system without specifying the PID version.

FGCA12. The existing Configuration Admin interfaces SHOULD be updated to support versioned PIDs.

FGCA13. The Configuration Admin interfaces SHOULD be enhanced to support setting configuration deltas. This includes atomically setting deltas across configuration PIDs.

FGCA014. The solution MUST provide the configuration version on all newly defined callbacks.

FGCA015. The solution MUST define what configuration, or configurations, will be provided to a Configuration Target in the event of multiple versions matching a version range.

5 Technical Solution

The solution is described in three separate parts. Section 5.1 describes the technical design for configuration delta APIs. Section 5.2 describes how updates for multiple PIDs can be received in a single callback. Section 5.3 describes how configuration objects are versioned.

Although described separately, these features can be combined and used together by clients of compliant Configuration Admin Service implementations.

5.1 Configuration Deltas

The solution is based around a utility class, the ConfigurationTracker, that provides users of the Configuration Admin Service with delta information. The ConfigurationTracker is used by the Bundle developer instead of the ManagedService. However the ConfigurationTracker utilizes the ManagedService under the covers. The class can be used with existing Configuration Admin Service implementations.

5.1.1 ConfigurationTracker class

The ConfigurationTracker class has the following API:

```
public class ConfigurationTracker {
```

Copyright © Progress Software Corporation 2010 All Rights Reserved

```
    public ConfigurationTracker(BundleContext context, String ... pids) {
        // Constructor
    }

    public ConfigurationTracker(BundleContext context,
                                Dictionary props, String ... pids) {
        // Constructor taking additional properties for the ManagedService
    }

    public void open() {
        // open the tracker
    }

    public void close() {
        // close the tracker
    }

    protected void changed(Map<String, Delta> changes, boolean configProvided)
            throws ConfigurationException {}
protected void added(Map<String, Object> additions)
            throws ConfigurationException {}

    protected void changed(Map<String, Delta> changes)
            throws ConfigurationException {}

    protected void deleted(Map<String, Object> deletions)
            throws ConfigurationException {}

    protected void nothing() throws ConfigurationException {}

    public static final class Delta {
        private final Object oldValue;
        private final Object newValue;

        private Delta(Object o, Object n) {
            oldValue = o;
            newValue = n;
        }

        public Object getOldValue() {
            return oldValue;
        }

        public Object getNewValue() {
            return newValue;
        }

        // … other methods such as equals() and hashCode()
        @Override
        public String toString() {
            return "<" + oldValue + "," + newValue + ">";
        }
    }
}
```

Copyright © Progress Software Corporation 2010 All Rights Reserved

The following APIs are is intended to be overridden by a subclass:

```
    protected void added(Map<String, Object> additions)
    protected void changed(Map<String, Delta> changes)
    protected void deleted(Map<String, Object> deletions)
    protected void nothing()protected void changed(Map<String, Delta> changes,
boolean configProvided)
        throws ConfigurationException;
```

The subclass will receive appropriate callbacks whenever the underlying ManagedService receives an `updated()` callback. The `nothing()`configProvided callback will be made whenparameter will be set to false when `updated(null)` is invoked. A typical use of the **`ConfigurationTracker`** is as follows:

```
    ConfigurationTracker ct = new ConfigurationTracker(ctx, "test.a") {
        protected void changed(Map<String, Delta> changes, boolean configProvided)
            throws ConfigurationException {
            // … process the changes map
        }
        protected void added(Map<String, Object> additions) {
            System.out.println("Added configuration: " + additions);
        }

        protected void changed(Map<String, Delta> changes) {
            System.out.println("Changed configuration: " + changes);
        }

        protected void deleted(Map<String, Object> deletions) {
            System.out.println("Deleted configuration: " + deletions);
        }

        protected void nothing() {
            System.out.println("There is no configuration, please use defaults");
        }
    };
    ct.open();
```

Remark: like with the ServiceTracker, the ConfigurationTracker is enabled by calling open() on it.

After use, typically in `BundleActivator.stop()`, `ct.close()` is called.

5.1.2 Managed Service Factories

TODO

5.2 Managed Services receiving updates for multiple PIDs at once

A ManagedService can now be registered with an additional property (osgi.managed.service.updates) to indicate that it wishes to receive updates to multiple PIDs in a single callback rather than multiple `updated()` callbacks. This is only relevant to ManagedServices that have registered themselves with multiple PIDs and all the key/value pairs of the relevant PIDs will have been merged into a single map when the `updated()` call is made. To receive multiple PIDs in a single callbacks, register the property osgi.managed.service.updates with the value '`mixed`' on the Service Registration of the ManagedService.

```
public class Activator implements BundleActivator {
```

Copyright © Progress Software Corporation 2010 All Rights Reserved

```
    public void start(BundleContext context) throws Exception {
        Hashtable props = new Hashtable();
        props.put(Constants.SERVICE_PID,
            new String [] {"org.example.mypid", "org.example.my_other_pid"});
        props.put("osgi.managed.service.updates", "mixed");
        ManagedService ms = new MyManagedService();
        context.registerService(ManagedService.class.getName(), ms, props);
    }
    //...
}
```

The following table defines the new ManagedService property.

Property key: osgi.managed.service.updates	
Data type: String+	
Description: This property specifies the mode in which the ManagedService is called back. Multiple values can be specified.	
value	***description***
mixed	Allow values of multiple PIDs in the updated() callbacks. The Constants.SERVICE_PID key lists the PIDs included in this callback. The configuration information regarding the multiple PIDs is merged into the single Dictionary provided in the update() callback. Specifying this value has no effect on ManagedServices which are only registered with one PID.

5.2.1 Checking Multi-PID Support at Runtime

Requesting multiple PIDs in Configuration Target callbacks is achieved by setting an additional property when the Configuration Target is registered in the Service Registry.

Since the ManagedService API itself hasn't changed, a bundle may wish to verify whether the a Configuration Admin implementation supports this behaviour.

The following options are available to achieve this:

1. Check that the org.osgi.service.cm package has a version of ~~2.0~~1.5 or higher by specifying this version on the Import-Package directive. This is a hard check and will not allow your ManagedService to work with pre 1.5~~x~~ Configuration Admin implementation.

2. If your ManagedService is capable of running with both a [1.0-1.4] ~~1.x~~ and a ~~2.x~~1.5 version of the Configuration Admin Service, it can find out how its being called back by checking the value of the osgi.managed.service.updates property on the Dictionary passed into the updated() callback. A older~~1.x~~ Configuration Admin Service will not set this property. A ~~2.x~~1.5 or newer Configuration Admin service will set the property to match the ManagedService registration value. If the property was not set on the ManagedService registration, a ~~2.x~~1.5 compliant service will the set .osgi.managed.service.updates="" on the updated() callbacks. The property starts with a dot '.' so is private.

5.2.2 Managed Service Factories

@@@ TODO

Copyright © Progress Software Corporation 2010 All Rights Reserved

5.2.3 Configuration Admin Service

In the OSGi 4.2 specifications, setting configuration values happens strictly by PID, through a series of calls that look like this:

```
ConfigurationAdmin cadmin = …
Configuration conf = cadmin.getConfiguration("pid1");
Dictionay d = … // dictionary of all values
conf.update(d);
```

This has the drawback that it doesn't allow setting configuration values for multiple Configuration PIDs in a single call. Being able to do this in a single call has the benefit that Configuration Targets that have registered themselves with multiple pids, can receive the changes in a single callback. This is required to address FGCA04.

Therefore the Configuration Admin Service is enhanced with an API to update Configuration information. This API allows the configuration values provided to span multiple Configuration PIDs.

The API will be added to the Configuration Admin interface, version 2.0.

```
public interface ConfigurationAdmin {
    // ... existing methods …

    /** Obtain a versioned Configuration object. */
    // embed the version in the PID, e.g. a.b.c;version=1.0
    // allow a Configuration object (or Objects?) that satisfies multiple pids
    Configuration getConfiguration(String ... pid) throws IOException;
    ~~Configuration getConfiguration(String pid, Version version) throws IOException;~~
    ~~Configuration getConfiguration(String pid, String location,~~
                                     ~~Version version) throws IOException;~~

    /**
     * Provide Configuration values for one or more ~~PIDs~~Configuration Admin
objects.
     * The configuration objects being updated must previously have been
     * created with one of the getConfiguration() methods.
     * @param updates The updated configuration Dictionaries.
     * One Dictionary per Configuration Object.
     */
    void update(Map<Configuration, Dictionary> configs) throws IOException;

}
```

The new API is used as follows:

TODO update this example

```
ConfigurationAdmin2 cadmin = …
Map<String, Map<String, Object>> changes =
          new HashMap<String, Map<String,Object>>();
Map<String, Object> pid1Changes = new HashMap<String, Object>();
Map<String, Object> pid2Changes = new HashMap<String, Object>();
changes.put("pid1", pid1Changes);
changes.put("pid2", pid2Changes);
```

Copyright © Progress Software Corporation 2010 All Rights Reserved

```
pid1Changes.put("a", "new_a");
pid1Changes.put("a.b.c", null); // unset
pid2Changes.put("foo", "bar");

cadmin.update(changes);
```

5.3 Versioned PIDs

The solution introduces versioning to Configuration objects. This allows multiple versions of configuration data with the same PID to co-exist. The configuration PID plus its version uniquely identifies the Configuration object.

A Configuration object with an associated version can be created using one of the ConfigurationAdmin.getConfiguration() APIs by appending the PID with a version attribute, e.g.:

```
ConfigurationAdmin cadmin = …
Configuration conf = cadmin.getConfiguration("my.pid;version=1.3");
```

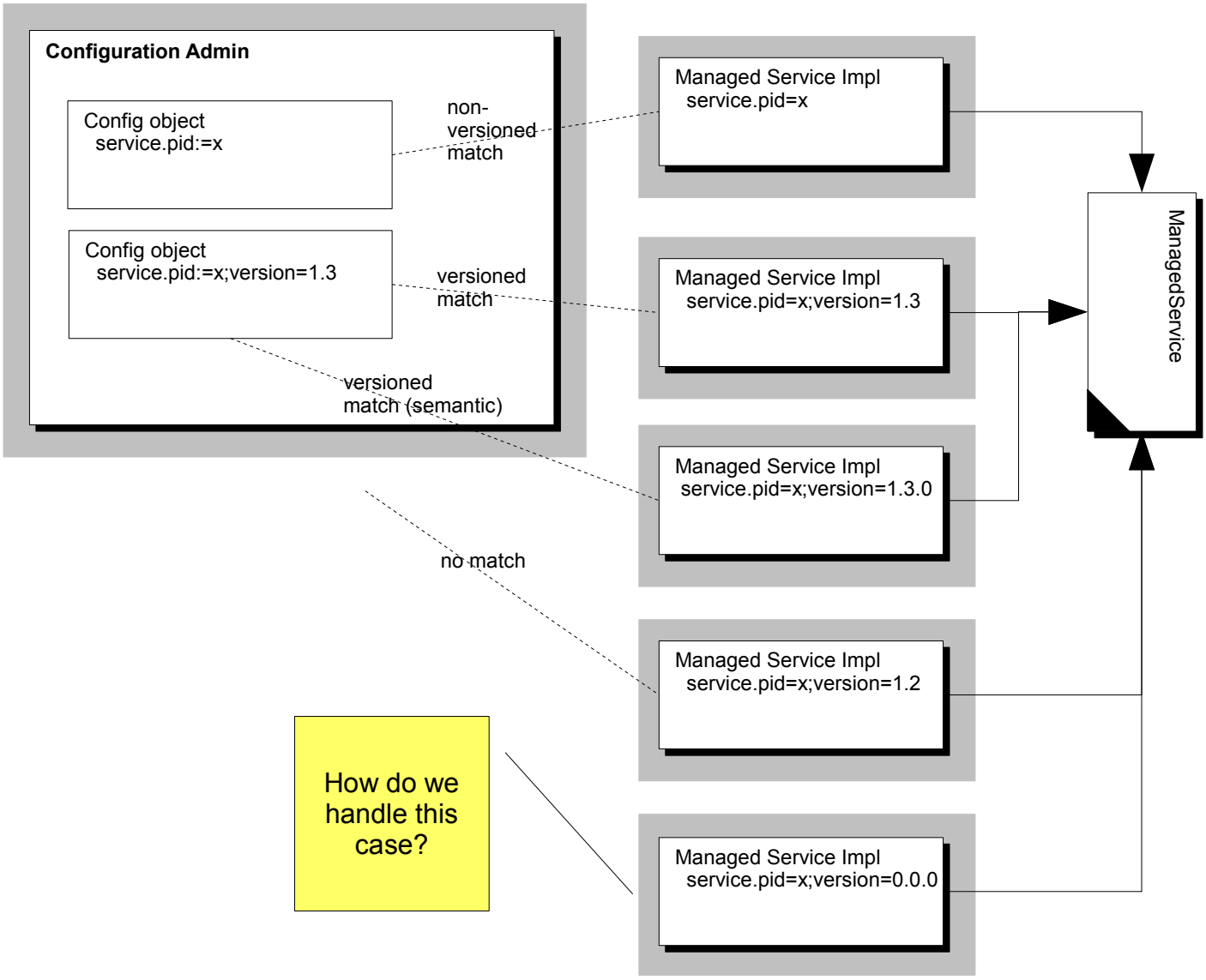
~~The Version of a Configuration object is provided in the API when creating a Configuration object. It is also present on the Dictionary with configuration information under the service.pid.version key. The value follows the standard OSGi version syntax and is available in the Dictionary as a String.~~

A ManagedService can indicate that it needs a particular version of the configuration information by appending a version attribute to the service.pid property. The Configuration Admin service implementation should take care of mapping service.pid=”my.pid;version=1” and service.pid=”my.pid;version=1.0.0” to the same configuration object.

~~specifying the service.pid.version property with its registration in the Service Registry. Not providing a version equals to specifying service.pid.version=”0.0.0”~~

The following diagram shows how multiple Configuration objects co-exist in the Configuration Admin Service and how version matching is done.

Copyright © Progress Software Corporation 2010 All Rights Reserved



5.3.1 ~~Creating a Versioned Configuration Object~~

~~A versioned Configuration object is created by obtaining it from the Configuration Admin service. Additional ConfigurationAdmin.getConfiguration() API variations now also accepts a version argument. The following example shows how to create or obtain a Configuration object with pid org.example.mypid and version 1.1:~~

```
ConfigurationAdmin cadmin = ...
Version v11 = new Version(1, 1, 0);
Configuration conf = cadmin.getConfiguration("org.example.mypid", v11);
conf.update(...);
```

~~The version is set in the configuration information under the key service.pid.version with a String value.~~

Copyright © Progress Software Corporation 2010 All Rights Reserved

5.3.2 Managed Service Factories

@@@ TODO

5.3.3 Registering a ManagedService for a Versioned PID

Obtaining a versioned configuration object is done by appending the version information to the Service PID when registering the Configuration Target in the Service Registry, for example:

```
public class Activator implements BundleActivator {
    public void start(BundleContext context) throws Exception {
        Hashtable props = new Hashtable();
        props.put(Constants.SERVICE_PID, "org.example.mypid;version=1.1");
        props.put(Constants.SERVICE_PID_VERSION, "1.1");
        ManagedService ms = new MyManagedService();
        context.registerService(ManagedService.class.getName(), ms, props);
    }
    //...
}
```

This will bind the ManagedService to a configuration object with PID org.example.mypid and version 1.1.

5.3.4 Backward Compatibility

When a version is omitted on a Configuration PID, this is understood to mean version=0.0.0. This will ensure that existing code will continue to work while avoiding clashes with code that depends on the newly introduced versioning functionality.

5.4 Configuration Plugins

TODO

6 Considered Alternatives

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

6.1 ManagedService2

There will be a new interface for ManagedServices that can be implemented to receive callbacks that only include the elements that have changed. These changes can span multiple PIDs, so the a applicable PIDs are provided with the callback as well.

The new interface doesn't extend the existing ManagedService interface. This means that an implementation that is *only* interested in deltas doesn't have to receive the old updated(Dictionary d) callbacks. A Managed Service that wishes to receive both old-style and new-style callbacks can register an object in the Service Registry that both implements ManagedService and ManagedService2 [name open for discussion].

Copyright © Progress Software Corporation 2010 All Rights Reserved

```
public interface ManagedService2
{
    /**
     * Called back with the deltas.
     * @param deltas A map with the PID as key and a map of
     *   key-delta pairs as value.
     */
    void updated(Map<String, Map<String, Delta>> deltas)
        throws ConfigurationException;
}
```

The Delta interface is as follows:

```
public interface Delta {
    Object getOldValue(); // if null: added
    Object getNewValue(); // if null: deleted
}
```

The Delta interface is quite small, do we need it at all - anyone a better idea? We could make it an inner interface of ManagedService2...

Example usage. The following code sample illustrates the use of ManagedService2 and prints out the changes it receives per PID.

```
public class MyManagedService2 implements ManagedService2 {
    public void updated(Map<String, Map<String, Delta>> deltas)
            throws ConfigurationException {
        for (String pid : deltas.keySet()) {
            Map<String, Delta> changes = deltas.get(pid);
            System.out.println("Changes for pid: " + pid);
            for (String key : changes.keySet()) {
                Delta delta = changes.get(key);
                System.out.print("  " + key + ":");
                System.out.println(delta.getOldValue()
                        + " -> " + delta.getNewValue());
            }
        }
    }
}
```

The Configuration Admin Service ensures that all changes provided in a single ConfigurationAdmin2.update() call will be provided to all relevant Configuration Targets in a single callback. For example, if a ManagedService2 registers itself with two PIDs and ConfigurationAdmin2.update() is called with changes to both PIDs, then the ManagedService2 will receive all its changes in a single ManagedService2.updated() call.

6.2 Version Ranges with Versioned PIDs

A Configuration object can declare that it is compatible with other versions of that configuration object. The Configuration Targets always specify exactly one version of the Configuration object that they are connected to. When linking a Configuration object to a Configuration Target the Configuration Admin service first tries to find an exact match. Failing that it tries to find a Configuration object that is marked as being compatible with the requested version.

A Configuration object can be marked as compatible with another version by providing the compatible-versions attribute when obtaining the Configuration object from the Configuration Admin service. The following example shows the creation of a configuration object that is compatible with versions 1.0-1.1 (inclusive) and 0.9

Copyright © Progress Software Corporation 2010 All Rights Reserved

```
Configuration conf = cadmin.getConfiguration("org.example.mypid;version=1.1;" +
                     "compatible-versions=\"[1.0, 1.1],0.9\"");
```

The compatible versions are set in the configuration information under the key service.pid.compatible-versions with a String+ value.

This is a departure from the current behaviour of how compatible versions of packages are wired to importers, where the onus is on the importer to declare what it is compatible with. This generally involves predicting the future and adherence to specific version policy guidelines and is therefore not reliable. However, for a Configuration object to describe what past configuration objects it's compatible with doesn't involve any guesswork and is therefore a better option.

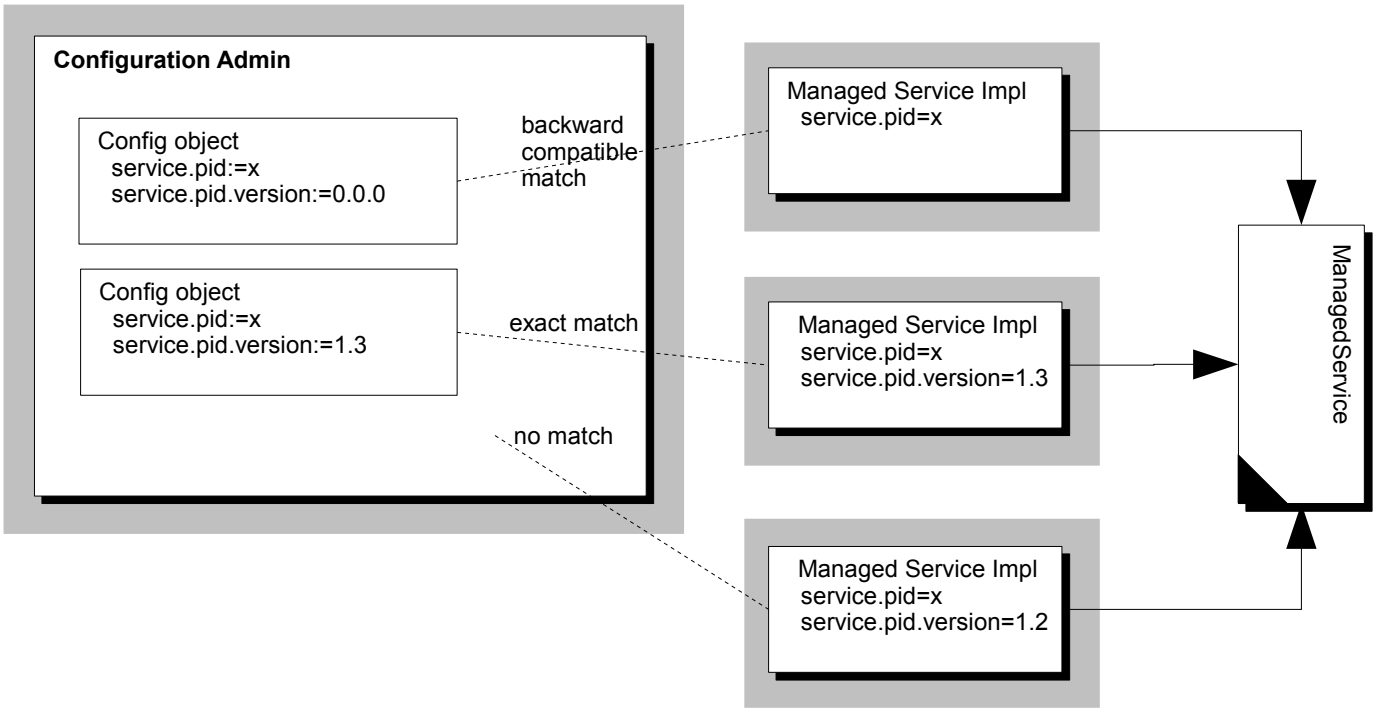
6.3 Version PIDs as separate service properties

The solution introduces versioning to Configuration objects. This allows multiple versions of configuration data with the same PID to co-exist. The configuration PID plus its version uniquely identifies the Configuration object.

The Version of a Configuration object is provided in the API when creating a Configuration object. It is also present on the Dictionary with configuration information under the service.pid.version key. The value follows the standard OSGi version syntax and is available in the Dictionary as a String.

A ManagedService can indicate that it needs a particular version of the configuration information by specifying the service.pid.version property with its registration in the Service Registry. Not providing a version equals to specifying service.pid.version="0.0.0"

The following diagram shows how multiple Configuration objects co-exist in the Configuration Admin Service and how version matching is done.



Copyright © Progress Software Corporation 2010 All Rights Reserved

6.3.1 Creating a Versioned Configuration Object

A versioned Configuration object is created by obtaining it from the Configuration Admin service. Additional ConfigurationAdmin.getConfiguration() API variations now also accepts a version argument. The following example shows how to create or obtain a Configuration object with pid org.example.mypid and version 1.1:

```
ConfigurationAdmin cadmin = ...
Version v11 = new Version(1, 1, 0);
Configuration conf = cadmin.getConfiguration("org.example.mypid", v11);
conf.update(...);
```

The version is set in the configuration information under the key service.pid.version with a String value.

6.3.2 Registering a ManagedService for a Versioned PID

Obtaining a versioned configuration object is done by appending the version information to the Service PID when registering the Configuration Target in the Service Registry, for example:

```
public class Activator implements BundleActivator {
    public void start(BundleContext context) throws Exception {
        Hashtable props = new Hashtable();
        props.put(Constants.SERVICE_PID, "org.example.mypid");
        props.put(Constants.SERVICE_PID_VERSION, "1.1");
        ManagedService ms = new MyManagedService();
        context.registerService(ManagedService.class.getName(), ms, props);
    }
    //...
}
```

~~@@@ put the old info in~~ @@@This will bind the ManagedService to a configuration object with PID org.example.mypid and version 1.1.

6.4 ManagedService receiving deltas

A ManagedService can now be registered with an additional property (osgi.managed.service.updates) to indicate that it will only receive deltas in its updates rather than the full map of values. To receive deltas in the callbacks, register the property osgi.managed.service.updates with the value 'deltas' on the Service Registration of the ManagedService.

```
public class Activator implements BundleActivator {
    public void start(BundleContext context) throws Exception {
        Hashtable props = new Hashtable();
        props.put(Constants.SERVICE_PID, "org.example.mypid");
        props.put("osgi.managed.service.updates", "deltas");
        ManagedService ms = new MyManagedService();
        context.registerService(ManagedService.class.getName(), ms, props);
    }
    //...
}
```

The following table defines the new ManagedService property.

Property key: osgi.managed.service.updates
Data type: String+

Copyright © Progress Software Corporation 2010 All Rights Reserved

Description: This property specifies the mode in which the ManagedService is called back. Multiple values can be specified, although some values are mutually exclusive.	
value	***description***
full (this is the default)	Pass the full collection of configuration information on the updated() callbacks. This behaviour is identical to the existing Configuration Admin behaviour. This value is mutually exclustive with 'deltas'.
deltas	Only provide deltas in the callbacks. This value is mutually exclusive with 'full'.

7 Security Considerations

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

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

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

8.2 Author's Address

Copyright © Progress Software Corporation 2010 All Rights Reserved

Name	David Bosschaert
Company	Progress Software
Address	The IONA Building, Shelbourne Road, Ballsbridge, Dublin 4, Ireland
Voice	+353 1 637 2000
e-mail	dbosscha@progress.com

Name	David Grigglestone
Company	Progress Software
Address	14 Oak Park Drive, Bedford, MA 01730, USA
Voice	+1 781-280-4000
e-mail	davidg@progress.com

Name	
Company	
Address	
Voice	
e-mail	

8.3 Acronyms and Abbreviations

8.4 End of Document

Copyright © Progress Software Corporation 2010 All Rights Reserved