RFC144: Configuration Admin Extension

Final

37 Pages

Abstract

This document proposes technical solution for fine grained access control to use Configuration Admin Service. To realize it, binding mechanism between bundles and Configuration objects and Configuration Permission are extended.

0 Document Information

0.1 Table of Contents

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 7.1.

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	*Nov 10, 2008*	Initial draft created. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp
2nd draft	*Dec 05, 2008*	Completely renewed based on the discussion on REG F2F in Nov. multiple binding is introduced. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp
3rd draft	*Jan 14, 2009*	Revised based on the discussion on CPEG F2F, Jan 13, 2009. All new methods introduced by 2nd draft are eliminated. Section 3.3 is added for terminologies. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
4th draft	*Jan 29, 2009*	Revised based on the discussion on REG F2F, Jan 23, 2009. Whole document is totally renewed. Simplify ConfigurationPermission extension, just added prefix match and new action “BOUND”. Binding mechanism has been changed. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
5th draft	*April 1, 2009*	Revised for wrong descriptions and add Use Case 4 and descriptions of considerations for the case that the Java Runtime Environment does not support permisssions. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
6th draft	*April 16, 2009*	Revised based on the discussion in REG F2F on April 2, 2009. Descriptions gets detailed. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
7th draft	*July 23, 2009*	Revised for clarifying descriptions. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
8th draft	*Sep 21, 2009*	Revised for clarifying descriptions. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp

Revision	Date	Comments
9th draft	*Oct 16, 2009*	Revised based on the discussion in Palo Alto CPEG F2F. Javadoc is newly added. Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
10th draft	*02/01/11*	Moved back from RFC 165 to implement the minimum necessary for REG – Peter Kriens
11th draft	*04/19/11*	Peter Kriens Update after Austin meeting: Now use a magic key ?. If a location starts with this question mark the location is intended for multiple delivery. This worked out very well and I like the fact that this now also works with security off. Security is now back to its old role of restricting and not widening. Defined update order to be service ranking. We decided that the MS(F) was not updated when it had no visibility to the configuration. However, have relaxed the rule for locations that start with ?. These MS(F) will be updated with null as if no configuration existed. Specified that if the location changes the CM must inform the targets appropriately. That is, delete configurations that are no longer visible and udpate the ones that become visible. Updated the Javadoc Specified that matching in the Configuration Permission is done with Filter substring matching
12th	*06/07/11*	Update from BJ's comments
Final	*6/9/2011*	Mark Final for CPEG voting.

1 Introduction

This RFC adds delivery of Configuration objects to multiple targets and extends the security model to allow checking of the location enabling grouping models.

2 Application Domain

- Configuration Admin delivers updates to only a single target and treats other targets as an error/warning.
- Currently Configuration Permission is limited to *, indicating that you either have permission to configure or not.

3 Problem Description

ConfigurationPermission introduced in R4.1 doesn't realize fine-grained control for access to Configuration objects. Current granularity of the permission allows independent bundles to interfere each others through changing configuration.

3.1 Binding between Configuration objects and bundles

In Configuration Admin Service Specification 1.2, one Configuration object is bound to either only one bundle or non. Therefore, common configuration information between multiple bundles must be set into multiple Configurations.

3.2 Coarse grain control for configuring Configuration objects

In Configuration Admin Service Specification 1.2, a bundle granted ConfigurationPermission can get Configuration objects bound to any bundles' locations and a bundle not granted ConfigurationPermission can get no Configuration objects except the ones bound to own bundle location. In other words, the following scenario is impossible to realize: bundle 1 is allowed to get Configuration objects of bundle1 and bundle 2, but not to get ones of bundle 3. In order to realize this, ConfigurationPermission must be changed to support a filter expression for its name as similarly as change of AdminPermission in Core Specification 1.3 (release 4.0).

3.3 Terminologies used in this document

Configuring bundle : a bundle which gets a Configuration object by calling a method of ConfigurationAdmin interface and sets a Dictionary object to the Configuration object by calling update() method of Confguration interface.

Target bundle : a bundle which registers a ManagedService or a ManagedFactoryService with a pid.

Configuration Manager(CM) : a bundle which implements Configuration Admin Service Spec. It registers Configuration Admin Service.

4 Requirements

[REQ01] New Spec MUST provide a solution to finer grained access control regarding getting Configuration objects.

[REQ02] New spec MUST allow one Configuration object to be bound to multiple bundles.

[REQ03] New spec MUST be backwards compatible.

5 Technical Solution

A normal Configuration object is only delivered to the target that has the given location. A null location can be used but this null location in the configuration is replaced with the first use of the configuration and the check takes place after this learning. The Configuration Permission to create and modify Configurations was Configuration[*,CONFIGURE]. Additionally, no security check was done if the manager was the bundle itself, that is, a bundle could always create and modify configurations for itself.

The first change for CM is that locations can now be delivered to multiple bundles that register an (MS(F) with the given PID. This is achieved by starting the location string with “?”. If the location string starts with this magic key, CM must update ALL MS(F) with the given PID regardless of their location. The remaining information after the prefix must be ignored when security is not on.

When security is on, the resource name of the Configuration Permission must be interpreted as the location, *including* the magic key. That is, Configuration Permission[“?http://www.aqute.biz/bundles/*”, CONFIGURE] allows its holder to maintain configurations for other bundles that have the given prefix for their location. Because the current API requires a configurer to have ConfigurationPermission[*,CONFIGURE], this change is fully backward compatible because * will match any location.

An additional action has been added, UPDATED. If the location of the configuration does not match the target's location, then the target must have an UPDATED permission where the resource name matches the configurations location. That is, if the location is “?com.acme”, the target MS(F) must have a Configuration Permisison like for example ConfigurationPermission[“?com.*”, UPDATED].

Location strings in the permissions can use the globbing rules from the Filter substring matching.

With security on:

- A configurer can be restricted to to only configure configurations for a given location with ConfigurationPermission[*name*,CONFIGURE]. Targets can only receive configurations if they have the proper ConfigurationPermission[*name*,UPDATED]. The consequence is that a deployer can configure the permissions of the system in such a way that a configurer can only provide configurations to a selected set of bundles.

- A configurer can be restricted them from delivering to multiple bundles by not including the magic prefix in the permission resource *name*.

- A grouping mechanism can now be devised when the configurer and targets agree on a naming scheme. For example, a configurer creates a Configuration for PID X with location “?com.acme”. The magic prefix indicates multiple delivery. A number of bundles register a Managed Service with PID X. The cm will require from the configurer ConfigurationPermission[?com.acme,CONFIGURE]. Any target bundle that has ConfigurationPermission[?com.acme,UPDATED] will be able to see the configuration.

If the Configuration has a null location the first found target must force the location to the target's bundle location before the permissions are checked and any subsequent target is addressed. Setting the location to null will restart this process. Setting the location to null requires ConfigurationPermission[*,CONFIGURE].

Currently, CM must unset the location if the location is learned and the target bundle is uninstalled. This remains so, however, a manager must be careful to not make this cause the configuration to leak. This can be achieved by explicitly setting the location by the configurer.

If multiple targets have to be called back then the callback order must be in service ranking order.

If the location is changed, then any targets that no longer have access to the configuration must be called with null (MS) or deleted (MSF). New targets that can now receive the configuration must be called back. Without security, the only case is when the location changes its multiple delivery states, with security the permission must be used to calculate the change set.

5.1 Changes Summary

- Locations starting with ? must be delivered to all targets with a corresponding PID

- Configuration Permission resource name must now match the location according to Filter substring matching rules.

- An UPDATED action is added. Targets that are updated must have this permission for the configuration's location unless they the location matches their own location.

- A target that has no visibility of a configuration due to security must be updated with null when it registers or the visibility had changed and the corresponding location starts with ?. If the location does not start with ? the target is not updated when the location does not match.

5.1.1 Backward Compatibility

The API is fully backward compatible as long as locations does not start with a “?”, which is a very unlikely prefix for a location string because these are normally URLs.

6 API

OSGi Javadoc

4/19/11 2:04 PM

Package Summary		Page
org.osgi.service.cm	Configuration Admin Package Version 1.4.	9

Package org.osgi.service.cm

Configuration Admin Package Version 1.4.

See:
[Description](#)

Interface Summary		Page
Configuration	The configuration information for a `ManagedService` or `ManagedServiceFactory` object.	10
ConfigurationAdmin	Service for administering configuration data.	15
ConfigurationListener	Listener for Configuration Events.	26
ConfigurationPlugin	A service interface for processing configuration dictionary before the update.	30
ManagedService	A service that can receive configuration data from a Configuration Admin service.	32
ManagedServiceFactory	Manage multiple service instances.	34

Class Summary		Page
ConfigurationEvent	A Configuration Event.	20
ConfigurationPermission	Indicates a bundle's authority to configure bundles, be updated by configuration admin, read the configurations, and act as a plugin.	27

Exception Summary		Page
ConfigurationException	An `Exception` class to inform the Configuration Admin service of problems with configuration data.	23

Package org.osgi.service.cm Description

Configuration Admin Package Version 1.4.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.cm; version="[1.4,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.cm; version="[1.4,1.5)"
```

Interface Configuration

org.osgi.service.cm

`public interface` **`Configuration`**

The configuration information for a `ManagedService` or `ManagedServiceFactory` object. The Configuration Admin service uses this interface to represent the configuration information for a `ManagedService` or for a service instance of a `ManagedServiceFactory`.

A `Configuration` object contains a configuration dictionary and allows the properties to be updated via this object. Bundles wishing to receive configuration dictionaries do not need to use this class - they register a `ManagedService` or `ManagedServiceFactory`. Only administrative bundles, and bundles wishing to update their own configurations need to use this class.

The properties handled in this configuration have case insensitive `String` objects as keys. However, case must be preserved from the last set key/value.

A configuration can be *bound* to a specific bundle or to a group of bundles using the *location* In its simplest form the location is the location of the target bundle that registered a Managed Service or a Managed Service Factory. However, if the location starts with `?` then the location indicates multiple delivery. In such a case the configuration must be delivered to all targets. If security is on, the Configuration Permission can be used to restrict the targets that receive updates. First, the configurer is required to have a Configuration Permission where the resource name matches the location, including the `?` prefix, except when the configurer configures itself. Second, the Configuration Admin must only update a target when the configuration location matches the location of the target's bundle or the target bundle has a Configuration Permission with the action UPDATE and a resource name that matches the configuration location. The matching of the resource name to the location is done using the Filter's wildcard substring matching rules. Bundles can always create, manipulate, and be updated from configurations that have a location that matches their location.

If a configuration's location is `null`, it is not yet bound to a location. It will become bound to the location of the first bundle that registers a `ManagedService` or `ManagedServiceFactory` object with the corresponding PID.

The same `Configuration` object is used for configuring both a Managed Service Factory and a Managed Service. When it is important to differentiate between these two the term "factory configuration" is used.

Version:

$Id: aad307ee968271be88c264acfcca6f197974d913 $

Method Summary		Page
void	**delete**() Delete this `Configuration` object.	12
boolean	**equals**(Object other) Equality is defined to have equal PIDs Two Configuration objects are equal when their PIDs are equal.	13
String	**getBundleLocation**() Get the bundle location.	13
String	**getFactoryPid**() For a factory configuration return the PID of the corresponding Managed Service Factory, else return `null`.	12
String	**getPid**() Get the PID for this `Configuration` object.	11
Dictionary	**getProperties**() Return the properties of this `Configuration` object.	11
int	**hashCode**() Hash code is based on PID.	13

void	**setBundleLocation**(String location) Bind this Configuration object to the specified location.	13
void	**update**() Update the Configuration object with the current properties.	12
void	**update**(Dictionary properties) Update the properties of this Configuration object.	11

Method Detail

getPid

String **getPid**()

Get the PID for this Configuration object.

Returns:
the PID for this Configuration object.
Throws:
IllegalStateException - if this configuration has been deleted

---

getProperties

Dictionary **getProperties**()

Return the properties of this Configuration object. The Dictionary object returned is a private copy for the caller and may be changed without influencing the stored configuration. The keys in the returned dictionary are case insensitive and are always of type String.

If called just after the configuration is created and before update has been called, this method returns null.

Returns:
A private copy of the properties for the caller or null. These properties must not contain the "service.bundleLocation" property. The value of this property may be obtained from the getBundleLocation method.
Throws:
IllegalStateException - if this configuration has been deleted

---

update

void **update**(Dictionary properties)
throws IOException

Update the properties of this Configuration object. Stores the properties in persistent storage after adding or overwriting the following properties:

- "service.pid" : is set to be the PID of this configuration.
- "service.factoryPid" : if this is a factory configuration it is set to the factory PID else it is not set.

These system properties are all of type String.

If the corresponding Managed Service/Managed Service Factory is registered, its updated method must be called asynchronously. Else, this callback is delayed until aforementioned registration occurs.

Also initiates an asynchronous call to all ConfigurationListeners with a ConfigurationEvent.CM_UPDATED event.

Parameters:
`properties` - the new set of properties for this configuration
Throws:
`IOException` - if update cannot be made persistent
`IllegalArgumentException` - if the `Dictionary` object contains invalid configuration types or contains case variants of the same key name.
`IllegalStateException` - if this configuration has been deleted

---

delete

```
void delete()
     throws IOException
```

Delete this `Configuration` object. Removes this configuration object from the persistent store. Notify asynchronously the corresponding Managed Service or Managed Service Factory. A `ManagedService` object is notified by a call to its `updated` method with a `null` properties argument. A `ManagedServiceFactory` object is notified by a call to its `deleted` method.

Also initiates an asynchronous call to all `ConfigurationListener`s with a `ConfigurationEvent.CM_DELETED` event.

Throws:
`IOException` - If delete fails
`IllegalStateException` - if this configuration has been deleted

---

getFactoryPid

```
String getFactoryPid()
```

For a factory configuration return the PID of the corresponding Managed Service Factory, else return `null`.

Returns:
factory PID or `null`
Throws:
`IllegalStateException` - if this configuration has been deleted

---

update

```
void update()
     throws IOException
```

Update the `Configuration` object with the current properties. Initiate the `updated` callback to the Managed Service or Managed Service Factory with the current properties asynchronously.

This is the only way for a bundle that uses a Configuration Plugin service to initiate a callback. For example, when that bundle detects a change that requires an update of the Managed Service or Managed Service Factory via its `ConfigurationPlugin` object.

Throws:
`IOException` - if update cannot access the properties in persistent storage
`IllegalStateException` - if this configuration has been deleted
See Also:
`ConfigurationPlugin`

---

setBundleLocation

```
void setBundleLocation(String location)
```

Bind this `Configuration` object to the specified location. If the bundleLocation parameter is `null` then the `Configuration` object will not be bound to a location. It will be set to the bundle's location before the first time a Managed Service/Managed Service Factory receives this `Configuration` object via the updated method and before any plugins are called. The bundle location will be set persistently.

If the location starts with `?` then all targets registered with the given PID must be updated.

If the location is changed the existing targets must be informed. If they can no longer see the configuration the configuration must be deleted or updated with `null`. If the configuration becomes visible then they must be updated.

Parameters:
`location` - a location (can have wildcards) or `null`
Throws:
`IllegalStateException` - If this configuration has been deleted.
`SecurityException` - when the required permissions are not available

---

getBundleLocation

```
String getBundleLocation()
```

Get the bundle location. Returns the bundle location or group to which this configuration is bound, or `null` if it is not yet bound to a bundle location or group. If the location starts with `?` then the configuration is delivered to all targets and not restricted to a single bundle.

Returns:
location to which this configuration is bound, or `null`.
Throws:
`IllegalStateException` - If this `Configuration` object has been deleted.
`SecurityException` - when the required permission is not available

---

equals

```
boolean equals(Object other)
```

Equality is defined to have equal PIDs Two Configuration objects are equal when their PIDs are equal.

Overrides:
`equals` in class `Object`
Parameters:
`other` - `Configuration` object to compare against
Returns:
`true` if equal, `false` if not a `Configuration` object or one with a different PID.

---

hashCode

```
int hashCode()
```

Hash code is based on PID. The hash code for two Configuration objects must be the same when the Configuration PID's are the same.

Overrides:
`hashCode` in class `Object`

Returns:
hash code for this Configuration object

Interface ConfigurationAdmin

org.osgi.service.cm

`public interface` **`ConfigurationAdmin`**

Service for administering configuration data.

The main purpose of this interface is to store bundle configuration data persistently. This information is represented in `Configuration` objects. The actual configuration data is a `Dictionary` of properties inside a `Configuration` object.

There are two principally different ways to manage configurations. First there is the concept of a Managed Service, where configuration data is uniquely associated with an object registered with the service registry.

Next, there is the concept of a factory where the Configuration Admin service will maintain 0 or more `Configuration` objects for a Managed Service Factory that is registered with the Framework.

The first concept is intended for configuration data about "things/services" whose existence is defined externally, e.g. a specific printer. Factories are intended for "things/services" that can be created any number of times, e.g. a configuration for a DHCP server for different networks.

Bundles that require configuration should register a Managed Service or a Managed Service Factory in the service registry. A registration property named `service.pid` (persistent identifier or PID) must be used to identify this Managed Service or Managed Service Factory to the Configuration Admin service.

When the ConfigurationAdmin detects the registration of a Managed Service, it checks its persistent storage for a configuration object whose `service.pid` property matches the PID service property (`service.pid`) of the Managed Service. If found, it calls `ManagedService.updated()` method with the new properties. The implementation of a Configuration Admin service must run these call-backs asynchronously to allow proper synchronization.

When the Configuration Admin service detects a Managed Service Factory registration, it checks its storage for configuration objects whose `service.factoryPid` property matches the PID service property of the Managed Service Factory. For each such `Configuration` objects, it calls the `ManagedServiceFactory.updated` method asynchronously with the new properties. The calls to the `updated` method of a `ManagedServiceFactory` must be executed sequentially and not overlap in time.

In general, bundles having permission to use the Configuration Admin service can only access and modify their own configuration information. Accessing or modifying the configuration of other bundles requires `ConfigurationPermission[group,CONFIGURE]`, where group is a group name or the location of a bundle.

`Configuration` objects can be *bound* to a specified bundle location or to a group. If a location is not set, it will be learned the first time a target is registered. If the location is learned this way, the Configuration Admin service must detect if the bundle corresponding to the location is uninstalled. If this occurs, the `Configuration` object must be unbound, that is its location field is set back to `null`.

If target's bundle location matches the configuration location it is always updated.

If the location starts with `?` then it must be delivered to all targets registered with the given PID. If security is on, the target bundle must have Configuration Permission[location,UPDATED], where location matches given the configuration location with wildcards as in the Filter substring match. The security must be verified using the `org.osgi.framework.Bundle.hasPermission()` method.

If a target cannot be updated because the location does not match or it has no permission and security is active then the Configuration Admin service must not do the normal callback, and should log an error.

The method descriptions of this class refer to a concept of "the calling bundle". This is a loose way of referring to the bundle which obtained the Configuration Admin service from the service registry. Implementations of `ConfigurationAdmin` must use a `org.osgi.framework.ServiceFactory` to support this concept.

Version:

$Id: 4807bff3699036c5b9526bd5bb2343c64f5e7d2a $

Field Summary		Page
String	**SERVICE_BUNDLELOCATION** Configuration property naming the location of the bundle that is associated with a a Configuration object.	16
String	**SERVICE_FACTORYPID** Configuration property naming the Factory PID in the configuration dictionary.	16

Method Summary		Page
Configuration	**createFactoryConfiguration**(String factoryPid) Create a new factory Configuration object with a new PID.	16
Configuration	**createFactoryConfiguration**(String factoryPid, String location) Create a new factory Configuration object with a new PID.	17
Configuration	**getConfiguration**(String pid) Get an existing or new Configuration object from the persistent store.	18
Configuration	**getConfiguration**(String pid, String location) Get an existing Configuration object from the persistent store, or create a new Configuration object.	17
Configuration[]	**listConfigurations**(String filter) List the current Configuration objects which match the filter.	18

Field Detail

SERVICE_FACTORYPID

```
public static final String SERVICE_FACTORYPID = "service.factoryPid"
```

Configuration property naming the Factory PID in the configuration dictionary. The property's value is of type String.

Since:
1.1

SERVICE_BUNDLELOCATION

```
public static final String SERVICE_BUNDLELOCATION = "service.bundleLocation"
```

Configuration property naming the location of the bundle that is associated with a a Configuration object. This property can be searched for but must not appear in the configuration dictionary for security reason. The property's value is of type String.

Since:
1.1

Method Detail

createFactoryConfiguration

```
Configuration createFactoryConfiguration(String factoryPid)
                                 throws IOException
```

Create a new factory `Configuration` object with a new PID. The properties of the new `Configuration` object are `null` until the first time that its `Configuration.update(Dictionary)` method is called.

It is not required that the `factoryPid` maps to a registered Managed Service Factory.

The `Configuration` object is bound to the location of the calling bundle. It is possible that the same factoryPid has associated configurations that are bound to different bundles. Bundles should only see the factory configurations that they are bound to or have the proper permission.

Parameters:
`factoryPid` - PID of factory (not `null`).
Returns:
A new `Configuration` object.
Throws:
`IOException` - if access to persistent storage fails.

---

createFactoryConfiguration

```
Configuration createFactoryConfiguration(String factoryPid,
                                         String location)
                                  throws IOException
```

Create a new factory `Configuration` object with a new PID. The properties of the new `Configuration` object are `null` until the first time that its `Configuration.update(Dictionary)` method is called.

It is not required that the `factoryPid` maps to a registered Managed Service Factory.

The `Configuration` is bound to the location specified. If this location is `null` it will be bound to the location of the first bundle that registers a Managed Service Factory with a corresponding PID. It is possible that the same factoryPid has associated configurations that are bound to different bundles. Bundles should only see the factory configurations that they are bound to or have the proper permission.

If the location starts with `?` then the configuration must be delivered to all targets with the corresponding PID.

Parameters:
`factoryPid` - PID of factory (not `null`).
`location` - A bundle location string, or `null`.
Returns:
a new `Configuration` object.
Throws:
`IOException` - if access to persistent storage fails.
`SecurityException` - when the require permissions are not available

---

getConfiguration

```
Configuration getConfiguration(String pid,
                               String location)
                        throws IOException
```

Get an existing `Configuration` object from the persistent store, or create a new `Configuration` object.

If a `Configuration` with this PID already exists in Configuration Admin service return it. The location parameter is ignored in this case.

Else, return a new `Configuration` object. This new object is bound to the location and the properties are set to `null`. If the location parameter is `null`, it will be set when a Managed Service with the corresponding PID is registered for the first time. If the location starts with `?` then the configuration is bound to all targets that are registered with the corresponding PID.

Parameters:
`pid` - Persistent identifier.
`location` - The bundle location string, or `null`.
Returns:
An existing or new `Configuration` object.
Throws:
`IOException` - if access to persistent storage fails.
`SecurityException` - when the require permissions are not available

---

getConfiguration

```
Configuration getConfiguration(String pid)
                        throws IOException
```

Get an existing or new `Configuration` object from the persistent store. If the `Configuration` object for this PID does not exist, create a new `Configuration` object for that PID, where properties are `null`. Bind its location to the calling bundle's location.

Otherwise, if the location of the existing `Configuration` object is `null`, set it to the calling bundle's location.

Parameters:
`pid` - persistent identifier.
Returns:
an existing or new `Configuration` matching the PID.
Throws:
`IOException` - if access to persistent storage fails.
`SecurityException` - when the required permission is not available

---

listConfigurations

```
Configuration[] listConfigurations(String filter)
                             throws IOException,
                                    org.osgi.framework.InvalidSyntaxException
```

List the current `Configuration` objects which match the filter.

Only `Configuration` objects with non- `null` properties are considered current. That is, `Configuration.getProperties()` is guaranteed not to return `null` for each of the returned `Configuration` objects.

When there is no security on then all configurations can be returned. If security is on, the caller must have ConfigurationPermission[location, `ConfigurationPermission.CONFIGURE`] or ConfigurationPermission[location,`ConfigurationPermission.UPDATED`].

The syntax of the filter string is as defined in the `org.osgi.framework.Filter` class. The filter can test any configuration properties including the following:

- `service.pid`-`String`- the PID under which this is registered
- `service.factoryPid`-`String`- the factory if applicable
- `service.bundleLocation`-`String`- the bundle location

The filter can also be `null`, meaning that all `Configuration` objects should be returned.

Parameters:
`filter` - A filter string, or `null` to retrieve all `Configuration` objects.
Returns:
All matching `Configuration` objects, or `null` if there aren't any.

Throws:

`IOException` - if access to persistent storage fails

`org.osgi.framework.InvalidSyntaxException` - if the filter string is invalid

Class ConfigurationEvent

org.osgi.service.cm

```
java.lang.Object
  └ org.osgi.service.cm.ConfigurationEvent
```

```
public class ConfigurationEvent
extends Object
```

A Configuration Event.

`ConfigurationEvent` objects are delivered to all registered `ConfigurationListener` service objects. ConfigurationEvents must be asynchronously delivered in chronological order with respect to each listener.

A type code is used to identify the type of event. The following event types are defined:

- CM_UPDATED
- CM_DELETED

Additional event types may be defined in the future.

Security Considerations. `ConfigurationEvent` objects do not provide `Configuration` objects, so no sensitive configuration information is available from the event. If the listener wants to locate the `Configuration` object for the specified pid, it must use `ConfigurationAdmin`.

Since:
1.2
Version:
$Id: b0efbb26d23b0931d901cac9c16f0a49b06a530b $
See Also:
`ConfigurationListener`

Field Summary	Page
static int **CM_DELETED** A `Configuration` has been deleted.	21
static int **CM_UPDATED** A `Configuration` has been updated.	21

Constructor Summary	Page
ConfigurationEvent(org.osgi.framework.ServiceReference reference, int type, String factoryPid, String pid) Constructs a `ConfigurationEvent` object from the given `ServiceReference` object, event type, and pids.	21

Method Summary	Page
String **getFactoryPid**() Returns the factory pid of the associated configuration.	21
String **getPid**() Returns the pid of the associated configuration.	22
org.osgi.framework.ServiceReference **getReference**() Return the `ServiceReference` object of the Configuration Admin service that created this event.	22
int **getType**() Return the type of this event.	22

Field Detail

CM_UPDATED

```
public static final int CM_UPDATED = 1
```

A `Configuration` has been updated.

This `ConfigurationEvent` type that indicates that a `Configuration` object has been updated with new properties. An event is fired when a call to `Configuration.update(Dictionary)` successfully changes a configuration.

The value of `CM_UPDATED` is 1.

CM_DELETED

```
public static final int CM_DELETED = 2
```

A `Configuration` has been deleted.

This `ConfigurationEvent` type that indicates that a `Configuration` object has been deleted. An event is fired when a call to `Configuration.delete()` successfully deletes a configuration.

The value of `CM_DELETED` is 2.

Constructor Detail

ConfigurationEvent

```
public ConfigurationEvent(org.osgi.framework.ServiceReference reference,
                          int type,
                          String factoryPid,
                          String pid)
```

Constructs a `ConfigurationEvent` object from the given `ServiceReference` object, event type, and pids.

Parameters:

`reference` - The `ServiceReference` object of the Configuration Admin service that created this event.
`type` - The event type. See `getType()`.
`factoryPid` - The factory pid of the associated configuration if the target of the configuration is a ManagedServiceFactory. Otherwise `null` if the target of the configuration is a ManagedService.
`pid` - The pid of the associated configuration.

Method Detail

getFactoryPid

```
public String getFactoryPid()
```

Returns the factory pid of the associated configuration.

Returns:

Returns the factory pid of the associated configuration if the target of the configuration is a ManagedServiceFactory. Otherwise `null` if the target of the configuration is a ManagedService.

---

getPid

```
public String getPid()
```

Returns the pid of the associated configuration.

Returns:
Returns the pid of the associated configuration.

---

getType

```
public int getType()
```

Return the type of this event.

The type values are:

- CM_UPDATED
- CM_DELETED

Returns:
The type of this event.

---

getReference

```
public org.osgi.framework.ServiceReference getReference()
```

Return the `ServiceReference` object of the Configuration Admin service that created this event.

Returns:
The `ServiceReference` object for the Configuration Admin service that created this event.

Class ConfigurationException

org.osgi.service.cm

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └org.osgi.service.cm.ConfigurationException
```

All Implemented Interfaces:
Serializable

```
public class ConfigurationException
extends Exception
```

An `Exception` class to inform the Configuration Admin service of problems with configuration data.

Version:
$Id: c9fb6fbf0fb8fc75291a073348fa0f4e56f248a7 $

Constructor Summary	Page
`ConfigurationException(String property, String reason)` Create a `ConfigurationException` object.	23
`ConfigurationException(String property, String reason, Throwable cause)` Create a `ConfigurationException` object.	24

Method Summary		Page
`Throwable`	`getCause()` Returns the cause of this exception or `null` if no cause was set.	24
`String`	`getProperty()` Return the property name that caused the failure or null.	24
`String`	`getReason()` Return the reason for this exception.	24
`Throwable`	`initCause(Throwable cause)` Initializes the cause of this exception to the specified value.	24

Constructor Detail

ConfigurationException

```
public ConfigurationException(String property,
                              String reason)
```

Create a `ConfigurationException` object.

Parameters:
`property` - name of the property that caused the problem, `null` if no specific property was the cause
`reason` - reason for failure

ConfigurationException

```
public ConfigurationException(String property,
                              String reason,
                              Throwable cause)
```

Create a `ConfigurationException` object.

Parameters:
`property` - name of the property that caused the problem, `null` if no specific property was the cause
`reason` - reason for failure
`cause` - The cause of this exception.

Since:
1.2

Method Detail

getProperty

```
public String getProperty()
```

Return the property name that caused the failure or null.

Returns:
name of property or null if no specific property caused the problem

---

getReason

```
public String getReason()
```

Return the reason for this exception.

Returns:
reason of the failure

---

getCause

```
public Throwable getCause()
```

Returns the cause of this exception or `null` if no cause was set.

Overrides:
`getCause` in class `Throwable`

Returns:
The cause of this exception or `null` if no cause was set.

Since:
1.2

---

initCause

```
public Throwable initCause(Throwable cause)
```

Initializes the cause of this exception to the specified value.

Overrides:

`initCause` in class `Throwable`

Parameters:

`cause` - The cause of this exception.

Returns:

This exception.

Throws:

`IllegalArgumentException` - If the specified cause is this exception.

`IllegalStateException` - If the cause of this exception has already been set.

Since:

1.2

Interface ConfigurationListener

org.osgi.service.cm

`public interface` **`ConfigurationListener`**

Listener for Configuration Events. When a `ConfigurationEvent` is fired, it is asynchronously delivered to a `ConfigurationListener`.

`ConfigurationListener` objects are registered with the Framework service registry and are notified with a `ConfigurationEvent` object when an event is fired.

`ConfigurationListener` objects can inspect the received `ConfigurationEvent` object to determine its type, the pid of the `Configuration` object with which it is associated, and the Configuration Admin service that fired the event.

Security Considerations. Bundles wishing to monitor configuration events will require `ServicePermission[ConfigurationListener,REGISTER]` to register a `ConfigurationListener` service.

Since:
1.2

Version:
$Id: bc0872c4df2541cba4060a0036f8aeb24a608051 $

Method Summary		Page
void	**configurationEvent**(ConfigurationEvent event) Receives notification of a Configuration that has changed.	26

Method Detail

configurationEvent

`void` **`configurationEvent`**`(ConfigurationEvent event)`

Receives notification of a Configuration that has changed.

Parameters:
`event` - The `ConfigurationEvent`.

Class ConfigurationPermission

org.osgi.service.cm

```
java.lang.Object
  └java.security.Permission
      └java.security.BasicPermission
          └org.osgi.service.cm.ConfigurationPermission
```

All Implemented Interfaces:
Guard, Serializable

```
final public class ConfigurationPermission
extends BasicPermission
```

Indicates a bundle's authority to configure bundles, be updated by configuration admin, read the configurations, and act as a plugin.

Since:
1.2
Version:
$Id: 828f2cd386d15ca4a09fd4ff57b5b582056ca211 $
ThreadSafe

Field Summary		Page
static String	**CONFIGURE** Provides permission to create new configurations for other bundles as well as manipulate them.	28
static String	**UPDATED** The permission to be updated, that is, act as a Managed Service or Managed Service Factory.The action string "updated".	28

Constructor Summary	Page
ConfigurationPermission(String name, String actions) Create a new ConfigurationPermission.	28

Method Summary		Page
boolean	**equals**(Object obj) Determines the equality of two ConfigurationPermission objects.	28
String	**getActions**() Returns the canonical string representation of the ConfigurationPermission actions.	29
int	**hashCode**() Returns the hash code value for this object.	29
boolean	**implies**(Permission p) Determines if a ConfigurationPermission object "implies" the specified permission.	28
PermissionCollection	**newPermissionCollection**() Returns a new PermissionCollection object suitable for storing ConfigurationPermissions.	29

Field Detail

CONFIGURE

```
public static final String CONFIGURE = "configure"
```

Provides permission to create new configurations for other bundles as well as manipulate them. The action string "configure".

UPDATED

```
public static final String UPDATED = "updated"
```

The permission to be updated, that is, act as a Managed Service or Managed Service Factory.The action string "updated".

Constructor Detail

ConfigurationPermission

```
public ConfigurationPermission(String name,
                               String actions)
```

Create a new ConfigurationPermission. The resource name of this permission is the location string. Wildcards are allowed to implement a grouping concept.

For checking the permission a `null` may be passed as the resource name. This requires permission for the string "*".

Parameters:

`name` - Location string, wildcard characters ('*') is allowed as in Filter substring matching. name may be `null` for .

`actions` - Comma separated list of `CONFIGURE`, `UPDATED`.

Method Detail

implies

```
public boolean implies(Permission p)
```

Determines if a `ConfigurationPermission` object "implies" the specified permission.

Overrides:

`implies` in class `BasicPermission`

Parameters:

`p` - The target permission to check.

Returns:

`true` if the specified permission is implied by this object; `false` otherwise.

---

equals

```
public boolean equals(Object obj)
```

Determines the equality of two `ConfigurationPermission` objects.

Two `ConfigurationPermission` objects are equal.

Overrides:
`equals` in class `BasicPermission`
Parameters:
`obj` - The object being compared for equality with this object.
Returns:
`true` if `obj` is equivalent to this `ConfigurationPermission`; `false` otherwise.

---

hashCode

```
public int hashCode()
```

Returns the hash code value for this object.

Overrides:
`hashCode` in class `BasicPermission`
Returns:
Hash code value for this object.

---

getActions

```
public String getActions()
```

Returns the canonical string representation of the `ConfigurationPermission` actions.

Always returns present `ConfigurationPermission` actions in the following order: `CONFIGURE`

Overrides:
`getActions` in class `BasicPermission`
Returns:
Canonical string representation of the `ConfigurationPermission` actions.

---

newPermissionCollection

```
public PermissionCollection newPermissionCollection()
```

Returns a new `PermissionCollection` object suitable for storing `ConfigurationPermission`s.

Overrides:
`newPermissionCollection` in class `BasicPermission`
Returns:
A new `PermissionCollection` object.

Interface ConfigurationPlugin

org.osgi.service.cm

```
public interface ConfigurationPlugin
```

A service interface for processing configuration dictionary before the update.

A bundle registers a `ConfigurationPlugin` object in order to process configuration updates before they reach the Managed Service or Managed Service Factory. The Configuration Admin service will detect registrations of Configuration Plugin services and must call these services every time before it calls the `ManagedService` or `ManagedServiceFactoryupdated` method. The Configuration Plugin service thus has the opportunity to view and modify the properties before they are passed to the Managed Service or Managed Service Factory.

Configuration Plugin (plugin) services have full read/write access to all configuration information that passes through them.

The `Integerservice.cmRanking` registration property may be specified. Not specifying this registration property, or setting it to something other than an `Integer`, is the same as setting it to the `Integer` zero. The `service.cmRanking` property determines the order in which plugins are invoked. Lower ranked plugins are called before higher ranked ones. In the event of more than one plugin having the same value of `service.cmRanking`, then the Configuration Admin service arbitrarily chooses the order in which they are called.

By convention, plugins with `service.cmRanking< 0` or `service.cmRanking > 1000` should not make modifications to the properties.

The Configuration Admin service has the right to hide properties from plugins, or to ignore some or all the changes that they make. This might be done for security reasons. Any such behavior is entirely implementation defined.

A plugin may optionally specify a `cm.target` registration property whose value is the PID of the Managed Service or Managed Service Factory whose configuration updates the plugin is intended to intercept. The plugin will then only be called with configuration updates that are targeted at the Managed Service or Managed Service Factory with the specified PID. Omitting the `cm.target` registration property means that the plugin is called for all configuration updates.

Version:
$Id: bb0b15c6deab98a81c44cf1781ed5924205d5bbd $

Field Summary		Page
String	**CM_RANKING** A service property to specify the order in which plugins are invoked.	31
String	**CM_TARGET** A service property to limit the Managed Service or Managed Service Factory configuration dictionaries a Configuration Plugin service receives.	30

Method Summary		Page
void	**modifyConfiguration**(org.osgi.framework.ServiceReference reference, Dictionary properties) View and possibly modify the a set of configuration properties before they are sent to the Managed Service or the Managed Service Factory.	31

Field Detail

CM_TARGET

```
public static final String CM_TARGET = "cm.target"
```

A service property to limit the Managed Service or Managed Service Factory configuration dictionaries a Configuration Plugin service receives. This property contains a `String[]` of PIDs. A Configuration Admin service must call a Configuration Plugin service only when this property is not set, or the target service's PID is listed in this property.

CM_RANKING

```
public static final String CM_RANKING = "service.cmRanking"
```

A service property to specify the order in which plugins are invoked. This property contains an `Integer` ranking of the plugin. Not specifying this registration property, or setting it to something other than an `Integer`, is the same as setting it to the `Integer` zero. This property determines the order in which plugins are invoked. Lower ranked plugins are called before higher ranked ones.

Since:
1.2

Method Detail

modifyConfiguration

```
void modifyConfiguration(org.osgi.framework.ServiceReference reference,
                         Dictionary properties)
```

View and possibly modify the a set of configuration properties before they are sent to the Managed Service or the Managed Service Factory. The Configuration Plugin services are called in increasing order of their `service.cmRanking` property. If this property is undefined or is a non- `Integer` type, 0 is used.

This method should not modify the properties unless the `service.cmRanking` of this plugin is in the range `0 <= service.cmRanking <= 1000`.

If this method throws any `Exception`, the Configuration Admin service must catch it and should log it.

A Configuration Plugin will only be called for properties from configurations that have a location for which the Configuration Plugin has permission when security is active. When security is not active, no filtering is done.

Parameters:
`reference` - reference to the Managed Service or Managed Service Factory
`properties` - The configuration properties. This argument must not contain the "service.bundleLocation" property. The value of this property may be obtained from the `Configuration.getBundleLocation` method.

Interface ManagedService

org.osgi.service.cm

`public interface` **`ManagedService`**

A service that can receive configuration data from a Configuration Admin service.

A Managed Service is a service that needs configuration data. Such an object should be registered with the Framework registry with the `service.pid` property set to some unique identifier called a PID.

If the Configuration Admin service has a `Configuration` object corresponding to this PID, it will callback the `updated()` method of the `ManagedService` object, passing the properties of that `Configuration` object.

If it has no such `Configuration` object, then it calls back with a `null` properties argument. Registering a Managed Service will always result in a callback to the `updated()` method provided the Configuration Admin service is, or becomes active. This callback must always be done asynchronously.

Else, every time that either of the `updated()` methods is called on that `Configuration` object, the `ManagedService.updated()` method with the new properties is called. If the `delete()` method is called on that `Configuration` object, `ManagedService.updated()` is called with a `null` for the properties parameter. All these callbacks must be done asynchronously.

The following example shows the code of a serial port that will create a port depending on configuration information.

```
class SerialPort implements ManagedService {

  ServiceRegistration registration;
  Hashtable configuration;
  CommPortIdentifier id;

  synchronized void open(CommPortIdentifier id,
  BundleContext context) {
    this.id = id;
    registration = context.registerService(
      ManagedService.class.getName(),
      this,
      getDefaults()
    );
  }

  Hashtable getDefaults() {
    Hashtable defaults = new Hashtable();
    defaults.put( "port", id.getName() );
    defaults.put( "product", "unknown" );
    defaults.put( "baud", "9600" );
    defaults.put( Constants.SERVICE_PID,
      "com.acme.serialport." + id.getName() );
    return defaults;
  }

  public synchronized void updated(
    Dictionary configuration  ) {
    if ( configuration ==  null )
      registration.setProperties( getDefaults() );
    else {
      setSpeed( configuration.get("baud") );
      registration.setProperties( configuration );
    }
  }
  ...
}
```

As a convention, it is recommended that when a Managed Service is updated, it should copy all the properties it does not recognize into the service registration properties. This will allow the Configuration Admin service to set properties on services which can then be used by other applications.

Normally, a single Managed Service for a given PID is given the configuration dictionary, this is the configuration that is bound to the location of the registering bundle. However, when security is on, a Managed Service can have Configuration Permission to also be updated for other locations.

Version:
$Id: ce1acb95f19a9ec6e09c5fcc80e5a68baf544c8c $

Method Summary		Page
void	**updated**(Dictionary properties) Update the configuration for a Managed Service.	33

Method Detail

updated

```
void updated(Dictionary properties)
      throws ConfigurationException
```

Update the configuration for a Managed Service.

When the implementation of `updated(Dictionary)` detects any kind of error in the configuration properties, it should create a new `ConfigurationException` which describes the problem. This can allow a management system to provide useful information to a human administrator.

If this method throws any other `Exception`, the Configuration Admin service must catch it and should log it.

The Configuration Admin service must call this method asynchronously with the method that initiated the callback. This implies that implementors of Managed Service can be assured that the callback will not take place during registration when they execute the registration in a synchronized method.

If the the location allows multiple managed services to be called back for a single configuration then the callbacks must occur in service ranking order. Changes in the location must be reflected by deleting the configuration if the configuration is no longer visible and updating when it becomes visible.

If no configuration exists for the corresponding PID, or the bundle has no access to this service, then the bundle must be called back with a `null` to signal that CM is active but there is no data.

Parameters:
`properties` - A copy of the Configuration properties, or `null` . This argument must not contain the "service.bundleLocation" property. The value of this property may be obtained from the `Configuration.getBundleLocation` method.

Throws:
`ConfigurationException` - when the update fails

Interface ManagedServiceFactory

org.osgi.service.cm

`public interface` **`ManagedServiceFactory`**

Manage multiple service instances. Bundles registering this interface are giving the Configuration Admin service the ability to create and configure a number of instances of a service that the implementing bundle can provide. For example, a bundle implementing a DHCP server could be instantiated multiple times for different interfaces using a factory.

Each of these *service instances* is represented, in the persistent storage of the Configuration Admin service, by a factory `Configuration` object that has a PID. When such a `Configuration` is updated, the Configuration Admin service calls the `ManagedServiceFactory` updated method with the new properties. When `updated` is called with a new PID, the Managed Service Factory should create a new factory instance based on these configuration properties. When called with a PID that it has seen before, it should update that existing service instance with the new configuration information.

In general it is expected that the implementation of this interface will maintain a data structure that maps PIDs to the factory instances that it has created. The semantics of a factory instance are defined by the Managed Service Factory. However, if the factory instance is registered as a service object with the service registry, its PID should match the PID of the corresponding `Configuration` object (but it should **not** be registered as a Managed Service!).

An example that demonstrates the use of a factory. It will create serial ports under command of the Configuration Admin service.

```
class SerialPortFactory
  implements ManagedServiceFactory {
  ServiceRegistration registration;
  Hashtable ports;
  void start(BundleContext context) {
    Hashtable properties = new Hashtable();
    properties.put( Constants.SERVICE_PID,
      "com.acme.serialportfactory" );
    registration = context.registerService(
      ManagedServiceFactory.class.getName(),
      this,
      properties
    );
  }
  public void updated( String pid,
    Dictionary properties  ) {
    String portName = (String) properties.get("port");
    SerialPortService port =
      (SerialPort) ports.get( pid );
    if ( port == null ) {
      port = new SerialPortService();
      ports.put( pid, port );
      port.open();
    }
    if ( port.getPortName().equals(portName) )
      return;
    port.setPortName( portName );
  }
  public void deleted( String pid ) {
    SerialPortService port =
      (SerialPort) ports.get( pid );
    port.close();
    ports.remove( pid );
  }
  ...
}
```

Version:
$Id: 5b78ae440baaf5f209422a8d8dbe502133a84128 $

Method Summary		Page
void	**deleted**(String pid) Remove a factory instance.	36
String	**getName**() Return a descriptive name of this factory.	35
void	**updated**(String pid, Dictionary properties) Create a new instance, or update the configuration of an existing instance.	35

Method Detail

getName

```
String getName()
```

Return a descriptive name of this factory.

Returns:
the name for the factory, which might be localized

---

updated

```
void updated(String pid,
             Dictionary properties)
      throws ConfigurationException
```

Create a new instance, or update the configuration of an existing instance. If the PID of the `Configuration` object is new for the Managed Service Factory, then create a new factory instance, using the configuration `properties` provided. Else, update the service instance with the provided `properties`.

If the factory instance is registered with the Framework, then the configuration `properties` should be copied to its registry properties. This is not mandatory and security sensitive properties should obviously not be copied.

If this method throws any `Exception`, the Configuration Admin service must catch it and should log it.

When the implementation of updated detects any kind of error in the configuration properties, it should create a new `ConfigurationException` which describes the problem.

The Configuration Admin service must call this method asynchronously. This implies that implementors of the `ManagedServiceFactory` class can be assured that the callback will not take place during registration when they execute the registration in a synchronized method.

If the security allows multiple managed service factories to be called back for a single configuration then the callbacks must occur in service ranking order.

It is valid to create multiple factory instances that are bound to different locations. Managed Service Factory services must only be updated with configurations that are bound to their location or that start with the `?` prefix and for which they have permission. Changes in the location must be reflected by deleting the corresponding configuration if the configuration is no longer visible or updating when it becomes visible.

Parameters:
`pid` - The PID for this configuration.
`properties` - A copy of the configuration properties. This argument must not contain the service.bundleLocation" property. The value of this property may be obtained from the `Configuration.getBundleLocation` method.

Throws:
`ConfigurationException` - when the configuration properties are invalid.

deleted

`void` **`deleted`**`(String pid)`

Remove a factory instance. Remove the factory instance associated with the PID. If the instance was registered with the service registry, it should be unregistered. The Configuration Admin must call deleted for each instance it received in `updated(String, Dictionary)`.

If this method throws any `Exception`, the Configuration Admin service must catch it and should log it.

The Configuration Admin service must call this method asynchronously.

Parameters:

`pid` - the PID of the service to be removed

7 Document Support

7.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. Koya, M. et al, RFP 95 Multiple Service Provider Separation.

[4]. Ikuo, Y, RFP110 Security Improvement for Configuration Admin Service.

7.2 Author's Address

Name	Ikuo YAMASAKI
Company	NTT Corporation
Address	Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan
Voice	+81-46-859-8537
e-mail	yamasaki.ikuo@lab.ntt.co.jp

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drezery, Beailieu 34160 France
Voice	+33633982260

7.3 Acronyms and Abbreviations

7.4 End of Document