OSGi Service Platform, Mobile Specification Release 4

OSGi Service Platform Mobile Specification

The OSGi Alliance

Release 4
August 2005

IOS Press
Ohmsha

Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

Table Of Contents

ISBN 1 58603 311 5 (IOS Press)
ISBN 4-274-90559-4 (Ohmsha)
Library of Congress Control Number: 2003107481

Publisher

IOS Press
Nieuwe Hemweg 6B
1013 BG Amsterdam
The Netherlands
fax: +31 20 620 3419
e-mail: order@iospress.nl

Distributor in the UK and Ireland
IOS Press/Lavis Marketing
73 Lime Walk
Headington
Oxford OX3 7AD
England
fax: +44 1865 75 0079

Distributor in the USA and Canada
IOS Press, Inc.
5795-G Burke Centre Parkway
Burke, VA 22015
USA
fax: +1 703 323 3668
e-mail: iosbooks@iospress.com

Distributor in Germany, Austria and Switzerland
IOS Press/LSL.de
Gerichtsweg 28
D-04103 Leipzig
Germany
fax: +49 341 995 4255

Distributor in Japan
Ohmsha, Ltd.
3-1 Kanda Nishiki-cho
Chiyoda-ku, Tokyo 101-8460
Japan
fax: +81 3 3233 2426

PRINTED IN THE NETHERLANDS

LEGAL TERMS AND CONDITIONS REGARDING SPECIFICATION

Implementation of certain elements of the Open Services Gateway Initiative (OSGi) Specification may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of OSGi). OSGi is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

THE RECIPIENT ACKNOWLEDGES AND AGREES THAT THE SPECIFICATION IS PROVIDED "AS IS" AND WITH NO WARRANTIES WHATSOEVER, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS OF ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. THE RECIPIENT'S USE OF THE SPECIFICATION IS SOLELY AT THE RECIPIENT'S OWN RISK. THE RECIPIENT'S USE OF THE SPECIFICATION IS SUBJECT TO THE RECIPIENT'S OSGi MEMBER AGREEMENT, IN THE EVENT THAT THE RECIPIENT IS AN OSGi MEMBER.

IN NO EVENT SHALL OSGi BE LIABLE OR OBLIGATED TO THE RECIPIENT OR ANY THIRD PARTY IN ANY MANNER FOR ANY SPECIAL, NON-COMPENSATORY, CONSEQUENTIAL, INDIRECT, INCIDENTAL, STATUTORY OR PUNITIVE DAMAGES OF ANY KIND, INCLUDING, WITHOUT LIMITATION, LOST PROFITS AND LOST REVENUE, REGARDLESS OF THE FORM OF ACTION, WHETHER IN CONTRACT, TORT, NEGLIGENCE, STRICT PRODUCT LIABILITY, OR OTHERWISE, EVEN IF OSGi HAS BEEN INFORMED OF OR IS AWARE OF THE POSSIBILITY OF ANY SUCH DAMAGES IN ADVANCE.

THE LIMITATIONS SET FORTH ABOVE SHALL BE DEEMED TO APPLY TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW AND NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDIES AVAILABLE TO THE RECIPIENT. THE RECIPIENT ACKNOWLEDGES AND AGREES THAT THE RECIPIENT HAS FULLY CONSIDERED THE FOREGOING ALLOCATION OF RISK AND FINDS IT REASONABLE, AND THAT THE FOREGOING LIMITATIONS ARE AN ESSENTIAL BASIS OF THE BARGAIN BETWEEN THE RECIPIENT AND OSGi.

IF THE RECIPIENT USES THE SPECIFICATION, THE RECIPIENT AGREES TO ALL OF THE FOREGOING TERMS AND CONDITIONS. IF THE RECIPIENT DOES NOT AGREE TO THESE TERMS AND CONDITIONS, THE RECIPIENT SHOULD NOT USE THE SPECIFICATION AND SHOULD CONTACT OSGi IMMEDIATELY.

Trademarks

OSGi™ is a trademark, registered trademark, or service mark of The Open Services Gateway Initiative in the US and other countries. Java is a trademark, registered trademark, or service mark of Sun Microsystems, Inc. in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

Feedback

This specification can be downloaded from the OSGi web site:
http:// www.osgi.org.

Comments about this specification can be mailed to:
speccomments@mail.osgi.org

OSGi Member Companies

4DHomeNet, Inc.	Acunia
Alpine Electronics Europe Gmbh	AMI-C
Atinav Inc.	BellSouth Telecommunications, Inc.
BMW	Bombardier Transportation
Cablevision Systems	Coactive Networks
Connected Systems, Inc.	Deutsche Telekom
Easenergy, Inc.	Echelon Corporation
Electricite de France (EDF)	Elisa Communications Corporation
Ericsson	Espial Group, Inc.
ETRI	France Telecom
Gatespace AB	Hewlett-Packard
IBM Corporation	ITP AS
Jentro AG	KDD R&D Laboratories Inc.
Legend Computer System Ltd.	Lucent Technologies
Metavector Technologies	Mitsubishi Electric Corporation
Motorola, Inc.	NTT
Object XP AG	On Technology UK, Ltd
Oracle Corporation	P&S Datacom Corporation
Panasonic	Patriot Scientific Corp. (PTSC)
Philips	ProSyst Software AG
Robert Bosch Gmbh	Samsung Electronics Co., LTD
Schneider Electric SA	Siemens VDO Automotive
Sharp Corporation	Sonera Corporation
Sprint Communications Company, L.P.	Sony Corporation
Sun Microsystems	TAC AB
Telcordia Technologies	Telefonica I+D
Telia Research	Texas Instruments, Inc.
Toshiba Corporation	Verizon
Whirlpool Corporation	Wind River Systems

OSGi Board and Officers

	Rafiul Ahad	VP of Product Development, Wireless and Voice Division, *Oracle*
VP Americas	Dan Bandera	Program Director & BLM for Client & OEM Technology, *IBM Corporation*
President	John R. Barr, Ph.D.	Director, Standards Realization, Corporate Offices, *Motorola, Inc.*
	Maurizio S. Beltrami	Technology Manager Interconnectivity, *Philips Consumer Electronics*
	Hans-Werner Bitzer M.A.	Head of Section Smart Home Products, *Deutsche Telekom AG*
	Steven Buytaert	Co-Founder and Co-CEO, *ACUNIA*
VP Asia Pacific	R. Lawrence Chan	Vice President Asia Pacific *Echelon Corporation*
CPEG chair	BJ Hargrave	OSGi Fellow and Senior Software Engineer, *IBM Corporation*
Technology Officer and editor		
	Peter Kriens	OSGi Fellow and CEO, *aQute*
Treasurer	Jeff Lund	Vice President, Business Development & Corporate Marketing , *Echelon Corporation*
Executive Director	Dave Marples	Vice President, *Global Inventures, Inc.*
	Hans-Ulrich Michel	Project Manager Information, Communication and Telematics, *BMW*
Secretary	Stan Moyer	Strategic Research Program Manager *Telcordia Technologies, Inc.*
	Behfar Razavi	Sr. Engineering Manager, Java Telematics Technology, *Sun Microsystems, Inc.*
VP Marketing	Susan Schwarze, PhD.	Marketing Director, *ProSyst*
VP Europe, Middle East and Africa		
	Staffan Truvé	Chairman, *Gatespace*

Foreword

John Barr, *President OSGi*

1 Introduction

The Open Services Gateway Initiative (OSGi™) was founded in March 1999. Its mission is to create open specifications for the network delivery of managed services to local networks and devices. The OSGi organization is the leading standard for next-generation Internet services to homes, cars, small offices, and other environments.

The OSGi service platform specification delivers an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a coordinated fashion. It enables an entirely new category of smart devices due to its flexible and managed deployment of services. The primary targets for the OSGi specifications are set top boxes, service gateways, cable modems, consumer electronics, PCs, industrial computers, cars and more. These devices that implement the OSGi specifications will enable service providers like telcos, cable operators, utilities, and others to deliver differentiated and valuable services over their networks.

This is the third release of the OSGi service platform specification developed by representatives from OSGi member companies. The OSGi Service Platform Release 3 mostly extends the existing APIs into new areas. The few modifications to existing APIs are backward compatible so that applications for previous releases should run unmodified on release 3 Frameworks. The built-in version management mechanisms allow bundles written for the new release to adapt to the old Framework implementations, if necessary.

1.1 Sections

1.2 What is New

1.3 Reader Level

This specification is written for the following audiences:

- Application developers
- Framework and system service developers (system developers)
- Architects

This specification assumes that the reader has at least one year of practical experience in writing Java programs. Experience with embedded systems and server environments is a plus. Application developers must be aware that the OSGi environment is significantly more dynamic than traditional desktop or server environments.

System developers require a *very* deep understanding of Java. At least three years of Java coding experience in a system environment is recommended. A Framework implementation will use areas of Java that are not normally encountered in traditional applications. Detailed understanding is required of class loaders, garbage collection, Java 2 security, and Java native library loading.

Architects should focus on the introduction of each subject. This introduction contains a general overview of the subject, the requirements that influenced its design, and a short description of its operation as well as the entities that are used. The introductory sections require knowledge of Java concepts like classes and interfaces, but should not require coding experience.

Most of these specifications are equally applicable to application developers and system developers.

1.4 Conventions and Terms

1.4.1 Typography

A fixed width, non-serif typeface (`sample`) indicates the term is a Java package, class, interface, or member name. Text written in this typeface is always related to coding.

Emphasis (*sample*) is used the first time an important concept is introduced.

When an example contains a line that must be broken over multiple lines, the « character is used. Spaces must be ignored in this case. For example:

```
http://www.acme.com/sp/ «
file?abc=12
```

is equivalent to:

```
http://www.acme.com/sp/file?abc=12
```

In many cases in these specifications, a syntax must be described. This syntax is based on the following symbols:

```
*             Repetition of the previous element zero or
              more times, e.g. ( ',' list ) *
+             Repetition one or more times
?             Previous element is optional
( ... )       Grouping
'...'         Literal
|             Or
[...]         Set (one of)
..            list, e.g. 1..5 is the list 1 2 3 4 5
<...>         Externally defined token
digit         ::= [0..9]
alpha         ::= [a..zA..Z]
token         ::= alpha(alpha|digit|'_'|'-')*
quoted-string ::= '"' ... '"'
jar-path      ::= file['/'file]
file          A valid file name in a zip file which
```

```
has no restrictions except that it may not
contain a '/'.
```

Spaces are ignored unless specifically noted.

1.4.2 Object Oriented Terminology

Concepts like classes, interfaces, objects, and services are distinct but subtly different. For example, "LogService" could mean an instance of the class LogService, could refer to the class LogService, or could indicate the functionality of the overall Log Service. Experts usually understand the meaning from the context, but this understanding requires mental effort. To highlight these subtle differences, the following conventions are used.

When the class is intended, its name is spelled exactly as in the Java source code and displayed in a fixed width typeface: for example the "HttpService class", "a method in HttpContext" or "a javax.servlet.Servlet object". A class name is fully qualified, like javax.servlet.Servlet, when the package is not obvious from the context nor is it in one of the well known java packages like java.lang, java.io, java.util and java.net. Otherwise, the package is omitted like in String.

Exception and permission classes are not followed by the word "object". Readability is improved when the "object" suffix is avoided. For example, "to throw a SecurityException" and to "to have FilePermission" instead of "to have a FilePermission object".

Permissions can further be qualified with their actions. ServicePermission[GET|REGISTER,com.acme.*] means a ServicePermission with the action GET and REGISTER for all service names starting with com.acme. A ServicePermission[REGISTER, Producer|Consumer] means the GET ServicePermission for the Producer or Consumer class.

When discussing functionality of a class rather than the implementation details, the class name is written as normal text. This convention is often used when discussing services. For example, "the User Admin service".

Some services have the word "Service" embedded in their class name. In those cases, the word "service" is only used once but is written with an upper case S. For example, "the Log Service performs".

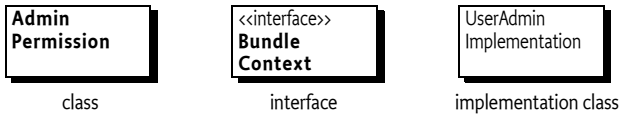
Service objects are registered with the OSGi Framework. Registration consists of the service object, a set of properties, and a list of classes and interfaces implemented by this service object. The classes and interfaces are used for type safety *and* naming. Therefore, it is said that a service object is registered *under* a class/interface. For example, "This service object is registered under PermissionAdmin."

1.4.3 Diagrams

The diagrams in this document illustrate the specification and are not normative. Their purpose is to provide a high-level overview on a single page. The following paragraphs describe the symbols and conventions used in these diagrams.

Classes or interfaces are depicted as rectangles, as in Figure 1. Interfaces are indicated with the qualifier ‹‹interface›› as the first line. The name of the class/interface is indicated in bold when it is part of the specification. Implementation classes are sometimes shown to demonstrate a possible implementation. Implementation class names are shown in plain text. In certain cases class names are abbreviated. This is indicated by ending the abbreviation with a period.

Figure 1 *Class and interface symbol*

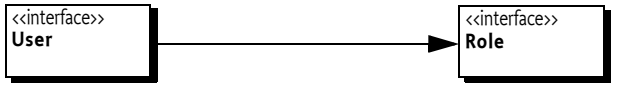


If an interface or class is used as a service object, it will have a black triangle in the bottom right corner.

Figure 2 *Service symbol*

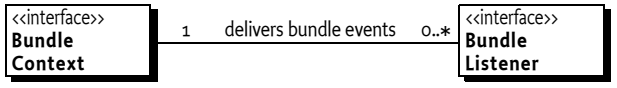
Inheritance (the extends or implements keyword in Java class definitions) is indicated with an arrow. Figure 3 shows that User implements or extends Role.

Figure 3 *Inheritance (implements or extends) symbol*



Relations are depicted with a line. The cardinality of the relation is given explicitly when relevant. Figure 4 shows that each (1) BundleContext object is related to 0 or more BundleListener objects, and that each BundleListener object is related to a single BundleContext object. Relations usually have some description associated with them. This description should be read from left to right and top to bottom, and includes the classes on both sides. For example: “A BundleContext object delivers bundle events to zero or more BundleListener objects.”

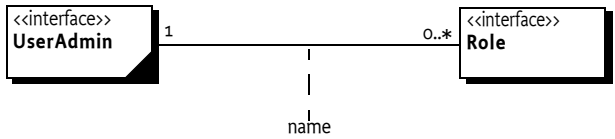
Figure 4 *Relations symbol*



Associations are depicted with a dashed line. Associations are between classes, and an association can be placed on a relation. For example, “every ServiceRegistration object has an associated ServiceReference object.” This association does not have to be a hard relationship, but could be derived in some way.

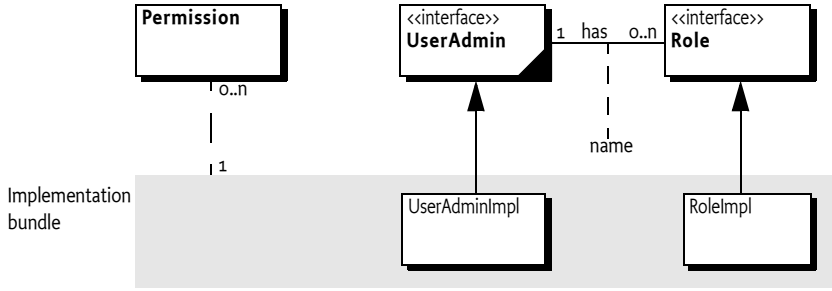
When a relationship is qualified by a name or an object, it is indicated by drawing a dotted line perpendicular to the relation and connecting this line to a class box or a description. Figure 5 shows that the relationship between a UserAdmin class and a Role class is qualified by a name. Such an association is usually implemented with a Dictionary object.

Figure 5 *Associations symbol*



Bundles are entities that are visible in normal application programming. For example, when a bundle is stopped, all its services will be unregistered. Therefore, the classes/interfaces that are grouped in bundles are shown on a grey rectangle.

Figure 6 *Bundles*



1.4.4 Key Words

This specification consistently uses the words *may*, *should*, and *must*. Their meaning is well defined in [1] *Bradner, S., Key words for use in RFCs to Indicate Requirement Levels.* A summary follows.

- *must* – An absolute requirement. Both the Framework implementation and bundles have obligations that are required to be fulfilled to conform to this specification.
- *should* – Recommended. It is strongly recommended to follow the description, but reasons may exist to deviate from this recommendation.
- *may* – Optional. Implementations must still be interoperable when these items are not implemented.

1.5 The Specification Process

Within the OSGi, specifications are developed by Expert Groups (EG). If a member company wants to participate in an EG, it must sign a Statement Of Work (SOW). The purpose of an SOW is to clarify the legal status of the material discussed in the EG. An EG will discuss material which already has

Intellectual Property (IP) rights associated with it, and may also generate new IP rights. The SOW, in conjunction with the member agreement, clearly defines the rights and obligations related to IP rights of the participants and other OSGi members.

To initiate work on a specification, a member company first submits a request for a proposal. This request is reviewed by the Market Requirement Committee which can either submit it to the Technical Steering Committee (TSC) or reject it. The TSC subsequently assigns the request to an EG to be implemented.

The EG will draft a number of proposals that meet the requirements from the request. Proposals usually contain Java code defining the API and semantics of the services under consideration. When the EG is satisfied with a proposal, it votes on it.

To assure that specifications can be implemented, reference implementations are created to implement the proposal. Test suites are also developed, usually by a different member company, to verify that the reference implementation (and future implementations by OSGi member companies) fulfill the requirements of the specifications. Reference implementations and test suites are *only* available to member companies.

Specifications combine a number of proposals to form a single document. The proposals are edited to form a set of consistent specifications, which are voted upon again by the EG. The specification is then submitted to all the member companies for review. During this review period, member companies must disclose any IP claims they have on the specification. After this period, the OSGi board of directors publishes the specification.

This Service Platform Release 3 specification was developed by the Core Platform Expert Group (CPEG), Device Expert Group (DEG), Remote Management Expert Group (RMEG), and Vehicle Expert Group (VEG).

1.6 Version Information

This document specifies OSGi Service Platform Release 3. This specification is backward compatible to releases 1 and 2.

New for this specification are:

- Wire Admin service
- Measurement utility
- Start Level service
- Execution Environments
- URL Stream and Content Handling
- Dynamic Import
- Position utility
- IO service
- XML service
- Jini service
- UPnP service
- OSGi Name-space
- Initial Provisioning service

Components in this specification have their own specification-version, independent of the OSGi Service Platform, Release 3 specification. The following table summarizes the packages and specification-versions for the different subjects.

Item	Package	Version
Framework	org.osgi.framework	1.2
Configuration Admin service	org.osgi.service.cm	1.1
Device Access	org.osgi.service.device	1.1
Http Service	org.osgi.service.http	1.1
IO Connector	org.osgi.service.io	1.0
Jini service	org.osgi.service.jini	1.0
Log Service	org.osgi.service.log	1.2
Metatype	org.osgi.service.metatype	1.0
Package Admin service	org.osgi.service.packageadmin	1.1
Permission Admin service	org.osgi.service.permissionadmin	1.1
Preferences Service	org.osgi.service.prefs	1.0
Initial Provisioning	org.osgi.service.provisioning	1.0
Bundle Start Levels	org.osgi.service.startlevel	1.0
Universal Plug & Play service	org.osgi.service.upnp	1.0
URL Stream and Content	org.osgi.service.url	1.0
User Admin service	org.osgi.service.useradmin	1.0
Wire Admin	org.osgi.service.wireadmin	1.0
Measurement utility	org.osgi.util.measurement	1.0
Position utility	org.osgi.util.position	1.0
Service Tracker	org.osgi.util.tracker	1.2
XML Parsers	org.osgi.util.xml	1.0

Table 1 *Packages and versions*

When a component is represented in a bundle, a specification-version is needed in the declaration of the Import-Package or Export-Package manifest headers. Package versioning is described in *Sharing Packages* on page 18.

1.7 Compliance Program

The OSGi offers a compliance program for the software product that includes an OSGi Framework and a set of zero or more core bundles collectively referred to as a Service Platform. Any services which exist in the org.osgi name-space and that are offered as part of a Service Platform must pass the conformance test suite in order for the product to be considered for inclusion in the compliance program. A Service Platform may be tested in

isolation and is independent of its host Virtual Machine. Certification means that a product has passed the conformance test suite(s) and meets certain criteria necessary for admission to the program, including the requirement for the supplier to warrant and represent that the product conforms to the applicable OSGi specifications, as defined in the compliance requirements.

The compliance program is a voluntary program and participation is the supplier's option. The onus is on the supplier to ensure ongoing compliance with the certification program and any changes which may cause this compliance to be brought into question should result in re-testing and re-submission of the Service Platform. Only members of the OSGi alliance are permitted to submit certification requests.

1.7.1 Compliance Claims.

In addition, any product that contains a certified OSGi Service Platform may be said to contain an *OSGi Compliant Service Platform*. The product itself is not compliant and should not be claimed as such.

More information about the OSGi Compliance program, including the process for inclusion and the list of currently certified products, can be found at http://www.osgi.org/compliance.

1.8 References

[1] *Bradner, S., Key words for use in RFCs to Indicate Requirement Levels* http://www.ietf.org/rfc/rfc2119.txt, March 1997.

[2] *OSGi Service Gateway Specification 1.0* http://www.osgi.org/resources/spec_download.asp

[3] *OSGi Service Platform, Release 2, October 2001* http://www.osgi.org/resources/spec_download.asp

2 Log Service Specification

Version 1.2

2.1 Introduction

The Log Service provides a general purpose message logger for the OSGi Service Platform. It consists of two services, one for logging information and another for retrieving current or previously recorded log information.

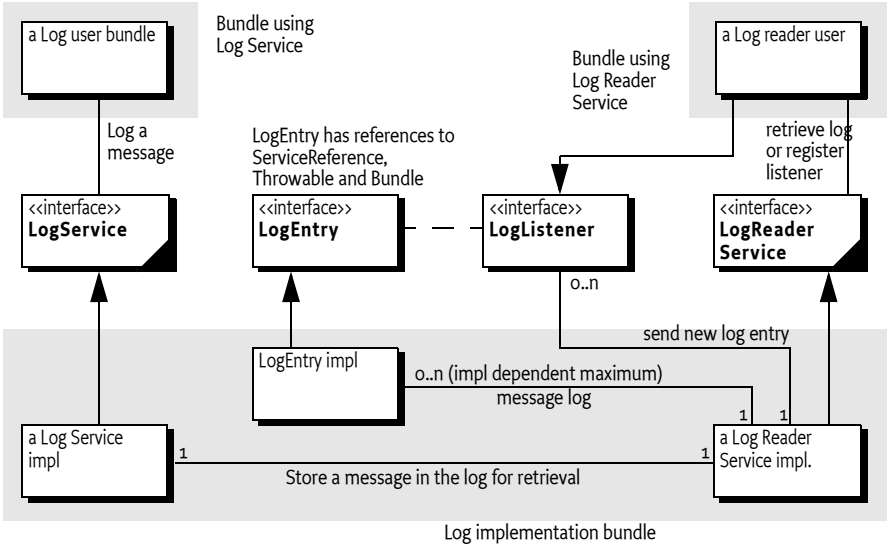
This specification defines the methods and semantics of interfaces which bundle developers can use to log entries and to retrieve log entries.

Bundles can use the Log Service to log information for the Operator. Other bundles, oriented toward management of the environment, can use the Log Reader Service to retrieve Log Entry objects that were recorded recently or to receive Log Entry objects as they are logged by other bundles.

2.1.1 Entities

- *LogService* – The service interface that allows a bundle to log information, including a message, a level, an exception, a `ServiceReference` object, and a `Bundle` object.
- *LogEntry* - An interface that allows access to a log entry in the log. It includes all the information that can be logged through the Log Service and a time stamp.
- *LogReaderService* - A service interface that allows access to a list of recent `LogEntry` objects, and allows the registration of a `LogListener` object that receives `LogEntry` objects as they are created.
- *LogListener* - The interface for the listener to `LogEntry` objects. Must be registered with the Log Reader Service.

Figure 7 Log Service Class Diagram org.osgi.service.log package



2.2 The Log Service Interface

The LogService interface allows bundle developers to log messages that can be distributed to other bundles, which in turn can forward the logged entries to a file system, remote system, or some other destination.

The LogService interface allows the bundle developer to:

- Specify a message and/or exception to be logged.
- Supply a log level representing the severity of the message being logged. This should be one of the levels defined in the LogService interface but it may be any integer that is interpreted in a user-defined way.
- Specify the Service associated with the log requests.

By obtaining a LogService object from the Framework service registry, a bundle can start logging messages to the LogService object by calling one of the LogService methods. A Log Service object can log any message, but it is primarily intended for reporting events and error conditions.

The LogService interface defines these methods for logging messages:

- log(int, String) – This method logs a simple message at a given log level.
- log(int, String, Throwable) – This method logs a message with an exception at a given log level.
- log(ServiceReference, int, String) – This method logs a message associated with a specific service.
- log(ServiceReference, int, String, Throwable) – This method logs a message with an exception associated with a specific service.

While it is possible for a bundle to call one of the log methods without providing a ServiceReference object, it is recommended that the caller supply the ServiceReference argument whenever appropriate, because it provides important context information to the operator in the event of problems.

The following example demonstrates the use of a log method to write a message into the log.

```
logService.log(
   myServiceReference,
   LogService.LOG_INFO,
   "myService is up and running"
);
```

In the example, the myServiceReference parameter identifies the service associated with the log request. The specified level, LogService.LOG_INFO, indicates that this message is informational.

The following example code records error conditions as log messages.

```
try {
   FileInputStream fis = new FileInputStream("myFile");
   int b;
   while ( (b = fis.read()) != -1 ) {
      ...
   }
   fis.close();
}
catch ( IOException exception ) {
   logService.log(
      myServiceReference,
      LogService.LOG_ERROR,
      "Cannot access file",
      exception );
}
```

Notice that in addition to the error message, the exception itself is also logged. Providing this information can significantly simplify problem determination by the Operator.

2.3 Log Level and Error Severity

The log methods expect a log level indicating error severity, which can be used to filter log messages when they are retrieved. The severity levels are defined in the LogService interface.

Callers must supply the log levels that they deem appropriate when making log requests. The following table lists the log levels.

Level	Descriptions
LOG_DEBUG	Used for problem determination and may be irrelevant to anyone but the bundle developer.
LOG_ERROR	Indicates the bundle or service may not be functional. Action should be taken to correct this situation.

Table 2 *Log Levels*

Level	Descriptions
LOG_INFO	May be the result of any change in the bundle or service and does not indicate a problem.
LOG_WARNING	Indicates a bundle or service is still functioning but may experience problems in the future because of the warning condition.

Table 2 *Log Levels*

2.4 Log Reader Service

The Log Reader Service maintains a list of LogEntry objects called the *log*. The Log Reader Service is a service that bundle developers can use to retrieve information contained in this log, and receive notifications about LogEntry objects when they are created through the Log Service.

The size of the log is implementation-specific, and it determines how far into the past the log entries go. Additionally, some log entries may not be recorded in the log in order to save space. In particular, LOG_DEBUG log entries may not be recorded. Note that this rule is implementation-dependent. Some implementations may allow a configurable policy to ignore certain LogEntry object types.

The LogReaderService interface defines these methods for retrieving log entries.

- getLog() – This method retrieves past log entries as an enumeration with the most recent entry first.
- addLogListener(LogListener) – This method is used to subscribe to the Log Reader Service in order to receive log messages as they occur. Unlike the previously recorded log entries, all log messages must be sent to subscribers of the Log Reader Service as they are recorded.
  A subscriber to the Log Reader Service must implement the LogListener interface.
  After a subscription to the Log Reader Service has been started, the subscriber's LogListener.logged method must be called with a LogEntry object for the message each time a message is logged.

The LogListener interface defines the following method:

- logged(LogEntry) – This method is called for each LogEntry object created. A Log Reader Service implementation must not filter entries to the LogListener interface as it is allowed to do for its log. A LogListener object should see all LogEntry objects that are created.

The delivery of LogEntry objects to the LogListener object should be done asynchronously.

2.5 Log Entry Interface

The LogEntry interface abstracts a log entry. It is a record of the information that was passed when an event was logged, and consists of a superset of information which can be passed through the LogService methods. The LogEntry interface defines these methods to retrieve information related to LogEntry objects:

- getBundle() – This method returns the Bundle object related to a LogEntry object.
- getException() – This method returns the exception related to a LogEntry object. In some implementations, the returned exception may not be the original exception. To avoid references to a bundle defined exception class, thus preventing an uninstalled bundle from being garbage collected, the Log Service may return an exception object of an implementation defined Throwable subclass. This object will attempt to return as much information as possible, such as the message and stack trace, from the original exception object .
- getLevel() – This method returns the severity level related to a LogEntry object.
- getMessage() – This method returns the message related to a LogEntry object.
- getServiceReference() –This method returns the ServiceReference object of the service related to a LogEntry object.
- getTime() – This method returns the time that the log entry was created.

2.6 Mapping of Events

Implementations of a Log Service must log Framework-generated events and map the information to LogEntry objects in a consistent way. Framework events must be treated exactly the same as other logged events and distributed to all LogListener objects that are associated with the Log Reader Service. The following sections define the mapping for the three different event types: Bundle, Service, and Framework.

2.6.1 Bundle Events Mapping

A Bundle Event is mapped to a LogEntry object according to Table 3, “Mapping of Bundle Events to Log Entries,” on page 15.

Log Entry method	Information about Bundle Event
getLevel()	LOG_INFO
getBundle()	Identifies the bundle to which the event happened. In other words, it identifies the bundle that was installed, started, stopped, updated, or uninstalled. This identification is obtained by calling getBundle() on the BundleEvent object.
getException()	null

Table 3 *Mapping of Bundle Events to Log Entries*

Log Entry method	Information about Bundle Event
getServiceReference()	null
getMessage()	The message depends on the event type: • INSTALLED – "BundleEvent INSTALLED" • STARTED – "BundleEvent STARTED" • STOPPED – "BundleEvent STOPPED" • UPDATED – "BundleEvent UPDATED" • UNINSTALLED – "BundleEvent UNINSTALLED"

Table 3 *Mapping of Bundle Events to Log Entries*

2.6.2 Service Events Mapping

A Service Event is mapped to a LogEntry object according to Table 4, "Mapping of Service Events to Log Entries," on page 16.

Log Entry method	Information about Service Event
getLevel()	LOG_INFO, except for the ServiceEvent.MODIFIED event. This event can happen frequently and contains relatively little information. It must be logged with a level of LOG_DEBUG.
getBundle()	Identifies the bundle that registered the service associated with this event. It is obtained by calling getServiceReference().getBundle() on the ServiceEvent object.
getException()	null
getServiceReference()	Identifies a reference to the service associated with the event. It is obtained by calling getServiceReference() on the ServiceEvent object.
getMessage()	This message depends on the actual event type. The messages are mapped as follows: • REGISTERED – "ServiceEvent REGISTERED" • MODIFIED – "ServiceEvent MODIFIED" • UNREGISTERING – "ServiceEvent UNREGISTERING"

Table 4 *Mapping of Service Events to Log Entries*

2.6.3 Framework Events Mapping

A Framework Event is mapped to a LogEntry object according to Table 5, "Mapping of Framework Event to Log Entries," on page 17.

Log Entry method	Information about Framework Event
getLevel()	LOG_INFO, except for the FrameworkEvent.ERROR event. This event represents an error and is logged with a level of LOG_ERROR.
getBundle()	Identifies the bundle associated with the event. This may be the system bundle. It is obtained by calling getBundle() on the FrameworkEvent object.
getException()	Identifies the exception associated with the error. This will be null for event types other than ERROR. It is obtained by calling getThrowable() on the FrameworkEvent object.
getServiceReference()	null
getMessage()	This message depends on the actual event type. The messages are mapped as follows: • STARTED – "FrameworkEvent STARTED" • ERROR – "FrameworkEvent ERROR" • PACKAGES_REFRESHED – "FrameworkEvent PACKAGES REFRESHED" • STARTLEVEL_CHANGED – "FrameworkEvent STARTLEVEL CHANGED"

Table 5 *Mapping of Framework Event to Log Entries*

2.7 Security

The Log Service should only be implemented by trusted bundles. This bundle requires ServicePermission[REGISTER,LogService|LogReaderService]. Virtually all bundles should get ServicePermission[GET,LogService]. The ServicePermission[GET,LogReaderService] should only be assigned to trusted bundles.

2.8 Changes

The following clarifications were made.

- The interpretation of the log level has been clarified to allow arbitrary integers.
- New Framework Event type strings are defined.
- LogEntry.getException is allowed to return a different exception object than the original exception object in order to allow garbage collection of the original object.
- The addLogListener method in the Log Reader Service no longer adds the same listener object twice.
- Delivery of Log Event objects to Log Listener objects must happen asynchronously. This delivery mode was undefined in previous releases.

2.9 org.osgi.service.log

The OSGi Log Service Package. Specification Version 1.2.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.log; specification-version=1.2
```

2.9.1 Summary

- LogEntry - Provides methods to access the information contained in an individual Log Service log entry. [p.11]
- LogListener - Subscribes to LogEntry objects from the LogReaderService. [p.11]
- LogReaderService - Provides methods to retrieve LogEntry objects from the log. [p.11]
- LogService - Provides methods for bundles to write messages to the log. [p.11]

2.9.2 public interface LogEntry

Provides methods to access the information contained in an individual Log Service log entry.

A LogEntry object may be acquired from the LogReaderService.getLog method or by registering a LogListener object.

See Also LogReaderService.getLog[p.20] , LogListener[p.11]

2.9.2.1 public Bundle getBundle( )

□ Returns the bundle that created this LogEntry object.

Returns The bundle that created this LogEntry object; null if no bundle is associated with this LogEntry object.

2.9.2.2 public Throwable getException( )

□ Returns the exception object associated with this LogEntry object.

In some implementations, the returned exception may not be the original exception. To avoid references to a bundle defined exception class, thus preventing an uninstalled bundle from being garbage collected, the Log Service may return an exception object of an implementation defined Throwable subclass. The returned object will attempt to provide as much information as possible from the original exception object such as the message and stack trace.

Returns Throwable object of the exception associated with this LogEntry;null if no exception is associated with this LogEntry object.

2.9.2.3 public int getLevel( )

□ Returns the severity level of this LogEntry object.

This is one of the severity levels defined by the LogService interface.

Returns Severity level of this LogEntry object.

See Also LogService.LOG_ERROR[p.21], LogService.LOG_WARNING[p.21], LogService.LOG_INFO[p.21], LogService.LOG_DEBUG[p.20]

2.9.2.4 public String getMessage( )

- Returns the human readable message associated with this LogEntry object.

Returns String containing the message associated with this LogEntry object.

2.9.2.5 public ServiceReference getServiceReference( )

- Returns the ServiceReference object for the service associated with this LogEntry object.

Returns ServiceReference object for the service associated with this LogEntry object; null if no ServiceReference object was provided.

2.9.2.6 public long getTime( )

- Returns the value of currentTimeMillis() at the time this LogEntry object was created.

Returns The system time in milliseconds when this LogEntry object was created.

See Also System.currentTimeMillis()

2.9.3 public interface LogListener extends EventListener

Subscribes to LogEntry objects from the LogReaderService.

A LogListener object may be registered with the Log Reader Service using the LogReaderService.addLogListener method. After the listener is registered, the logged method will be called for each LogEntry object created. The LogListener object may be unregistered by calling the LogReaderService.removeLogListener method.

See Also LogReaderService[p.11], LogEntry[p.11], LogReaderService.addLogListener(LogListener)[p.20], LogReaderService.removeLogListener(LogListener)[p.20]

2.9.3.1 public void logged(LogEntry entry)

entry A LogEntry object containing log information.

- Listener method called for each LogEntry object created.

  As with all event listeners, this method should return to its caller as soon as possible.

See Also LogEntry[p.11]

2.9.4 public interface LogReaderService

Provides methods to retrieve LogEntry objects from the log.

There are two ways to retrieve LogEntry objects:

- The primary way to retrieve LogEntry objects is to register a LogListener object whose LogListener.logged method will be called for each entry added to the log.
- To retrieve past LogEntry objects, the getLog method can be called which will return an Enumeration of all LogEntry objects in the log.

See Also LogEntry[p.11], LogListener[p.11], LogListener.logged(LogEntry)[p.19]

2.9.4.1 public void addLogListener(LogListener listener)

listener A LogListener object to register; the LogListener object is used to receive LogEntry objects.

- Subscribes to LogEntry objects.

  This method registers a LogListener object with the Log Reader Service. The LogListener.logged(LogEntry) method will be called for each LogEntry object placed into the log.

  When a bundle which registers a LogListener object is stopped or otherwise releases the Log Reader Service, the Log Reader Service must remove all of the bundle's listeners.

  If this Log Reader Service's list of listeners already contains a listener l such that (l==listener), this method does nothing.

See Also LogListener[p.11], LogEntry[p.11], LogListener.logged(LogEntry)[p.19]

2.9.4.2 public Enumeration getLog( )

- Returns an Enumeration of all LogEntry objects in the log.

  Each element of the enumeration is a LogEntry object, ordered with the most recent entry first. Whether the enumeration is of all LogEntry objects since the Log Service was started or some recent past is implementation-specific. Also implementation-specific is whether informational and debug LogEntry objects are included in the enumeration.

2.9.4.3 public void removeLogListener(LogListener listener)

listener A LogListener object to unregister.

- Unsubscribes to LogEntry objects.

  This method unregisters a LogListener object from the Log Reader Service.

  If listener is not contained in this Log Reader Service's list of listeners, this method does nothing.

See Also LogListener[p.11]

2.9.5 public interface LogService

Provides methods for bundles to write messages to the log.

LogService methods are provided to log messages; optionally with a ServiceReference object or an exception.

Bundles must log messages in the OSGi environment with a severity level according to the following hierarchy:

1 LOG_ERROR[p.21]
2 LOG_WARNING[p.21]
3 LOG_INFO[p.21]
4 LOG_DEBUG[p.20]

2.9.5.1 public static final int LOG_DEBUG = 4

A debugging message (Value 4).

This log entry is used for problem determination and may be irrelevant to anyone but the bundle developer.

2.9.5.2 public static final int LOG_ERROR = 1

An error message (Value 1).

This log entry indicates the bundle or service may not be functional.

2.9.5.3 public static final int LOG_INFO = 3

An informational message (Value 3).

This log entry may be the result of any change in the bundle or service and does not indicate a problem.

2.9.5.4 public static final int LOG_WARNING = 2

A warning message (Value 2).

This log entry indicates a bundle or service is still functioning but may experience problems in the future because of the warning condition.

2.9.5.5 public void log(int level, String message)

level The severity of the message. This should be one of the defined log levels but may be any integer that is interpreted in a user defined way.

message Human readable string describing the condition or `null`.

- Logs a message.

  The `ServiceReference` field and the `Throwable` field of the `LogEntry` object will be set to `null`.

See Also `LOG_ERROR`[p.21], `LOG_WARNING`[p.21], `LOG_INFO`[p.21], `LOG_DEBUG`[p.20]

2.9.5.6 public void log(int level, String message, Throwable exception)

level The severity of the message. This should be one of the defined log levels but may be any integer that is interpreted in a user defined way.

message The human readable string describing the condition or `null`.

exception The exception that reflects the condition or `null`.

- Logs a message with an exception.

  The `ServiceReference` field of the `LogEntry` object will be set to `null`.

See Also `LOG_ERROR`[p.21], `LOG_WARNING`[p.21], `LOG_INFO`[p.21], `LOG_DEBUG`[p.20]

2.9.5.7 public void log(ServiceReference sr, int level, String message)

sr The `ServiceReference` object of the service that this message is associated with or `null`.

level The severity of the message. This should be one of the defined log levels but may be any integer that is interpreted in a user defined way.

message Human readable string describing the condition or `null`.

- Logs a message associated with a specific ServiceReference object.

  The Throwable field of the LogEntry will be set to null.

See Also LOG_ERROR[p.21], LOG_WARNING[p.21], LOG_INFO[p.21], LOG_DEBUG[p.20]

2.9.5.8 public void log(ServiceReference sr, int level, String message, Throwable exception)

sr The ServiceReference object of the service that this message is associated with.

level The severity of the message. This should be one of the defined log levels but may be any integer that is interpreted in a user defined way.

message Human readable string describing the condition or null.

exception The exception that reflects the condition or null.

- Logs a message with an exception associated and a ServiceReference object.

See Also LOG_ERROR[p.21], LOG_WARNING[p.21], LOG_INFO[p.21], LOG_DEBUG[p.20]

3 Configuration Admin Service Specification

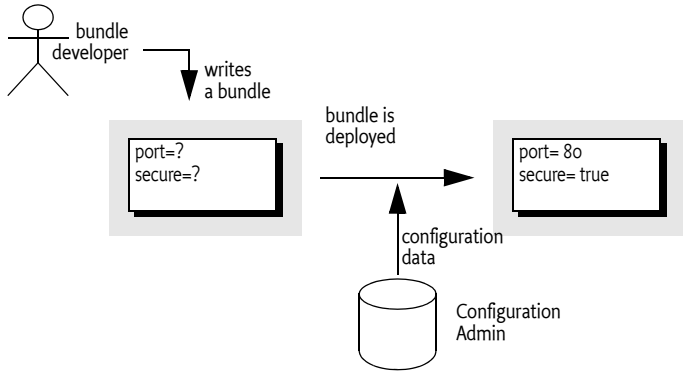
Version 1.1

3.1 Introduction

The Configuration Admin service is an important aspect of the deployment of an OSGi Service Platform. It allows an Operator to set the configuration information of deployed bundles.

Configuration is the process of defining the configuration data of bundles and assuring that those bundles receive that data when they are active in the OSGi Service Platform.

Figure 8 *Configuration Admin Service Overview*



3.1.1 Essentials

The following requirements and patterns are associated with the Configuration Admin service specification:

- *Local Configuration* – The Configuration Admin service must support bundles that have their own user interface to change their configurations.
- *Reflection* – The Configuration Admin service must be able to deduce the names and types of the needed configuration data.
- *Legacy* – The Configuration Admin service must support configuration data of existing entities (such as devices).
- *Object Oriented* – The Configuration Admin service must support the creation and deletion of instances of configuration information so that a bundle can create the appropriate number of services under the control of the Configuration Admin service.

- *Embedded Devices* – The Configuration Admin service must be deployable on a wide range of platforms. This requirement means that the interface should not assume file storage on the platform. The choice to use file storage should be left to the implementation of the Configuration Admin service.
- *Remote versus Local Management* – The Configuration Admin service must allow for a remotely managed OSGi Service Platform, and must not assume that configuration information is stored locally. Nor should it assume that the Configuration Admin service is always done remotely. Both implementation approaches should be viable.
- *Availability* – The OSGi environment is a dynamic environment that must run continuously (24/7/365). Configuration updates must happen dynamically and should not require restarting of the system or bundles.
- *Immediate Response* – Changes in configuration should be reflected immediately.
- *Execution Environment* – The Configuration Admin service will not require more than an environment that fulfills the minimal execution requirements.
- *Communications* – The Configuration Admin service should not assume "always-on" connectivity, so the API is also applicable for mobile applications in cars, phones, or boats.
- *Extendability* – The Configuration Admin service should expose the process of configuration to other bundles. This exposure should at a minimum encompass initiating an update, removing certain configuration properties, adding properties, and modifying the value of properties potentially based on existing property or service values.
- *Complexity Trade-offs* – Bundles in need of configuration data should have a simple way of obtaining it. Most bundles have this need and the code to accept this data. Additionally, updates should be simple from the perspective of the receiver.
  Trade-offs in simplicity should be made at the expense of the bundle implementing the Configuration Admin service and in favor of bundles that need configuration information. The reason for this choice is that normal bundles will outnumber Configuration Admin bundles.

3.1.2 Operation

This specification is based on the concept of a Configuration Admin service that manages the configuration of an OSGi Service Platform. It maintains a database of `Configuration` objects, locally or remote. This service monitors the service registry and provides configuration information to services that are registered with a `service.pid` property, the Persistent IDentity (PID), and implement one of the following interfaces:

- *Managed Service* – A service registered with this interface receives its *configuration dictionary* from the database or receives null when no such configuration exists or when an existing configuration has never been updated.
- *Managed Service Factory* – Services registered with this interface receive several configuration dictionaries when registered. The database contains zero or more configuration dictionaries for this service. Each configuration dictionary is given sequentially to the service.

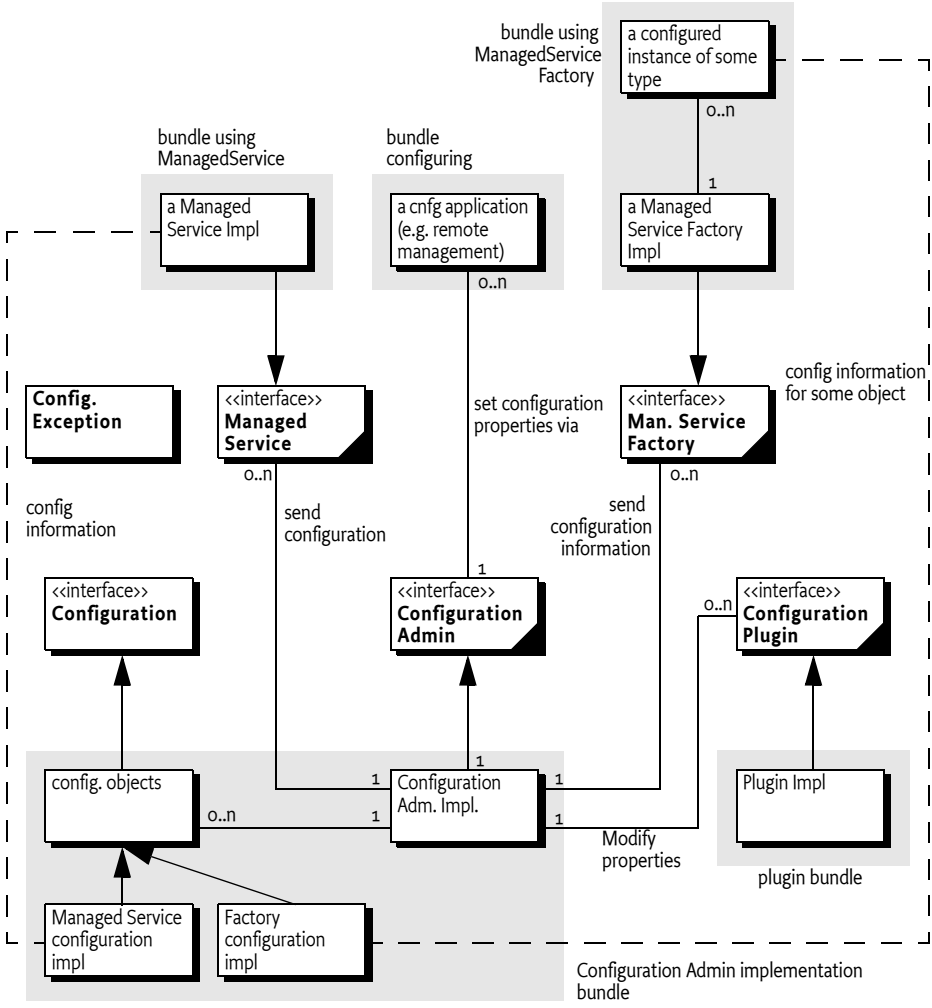
The database can be manipulated either by the Management Agent or bundles that configure themselves.

Other parties can provide Configuration Plugin services. Such services participate in the configuration process. They can inspect the configuration dictionary and modify it before it reaches the target service.

3.1.3 Entities

- *Configuration information* – The information needed by a bundle before it can provide its intended functionality.
- *Configuration dictionary* – The configuration information when it is passed to the target service. It consists of a Dictionary object with a number of properties and identifiers.
- *Configuring Bundle* – A bundle that modifies the configuration information through the Configuration Admin service. This bundle is either a management bundle or the bundle for which the configuration information is intended.
- *Configuration Target* – The target (bundle or service) that will receive the configuration information. For services, there are two types of targets: ManagedServiceFactory or ManagedService objects.
- *Configuration Admin Service* – This service is responsible for supplying configuration target bundles with their configuration information. It maintains a database with configuration information, keyed on the service.pid of configuration target services. These services receive their configuration dictionary or dictionaries when they are registered with the Framework. Configurations can be modified or extended using Configuration Plugin services before they reach the target bundle.
- *Managed Service* – A Managed Service represents a client of the Configuration Admin service, and is thus a configuration target. Bundles should register a Managed Service to receive the configuration data from the Configuration Admin service. A Managed Service adds a unique service.pid service registration property as a primary key for the configuration information.
- *Managed Service Factory* – A Managed Service Factory can receive a number of configuration dictionaries from the Configuration Admin service, and is thus also a configuration target service. It should register with a service.pid and receives zero or more configuration dictionaries. Each dictionary has its own PID.
- *Configuration Object* – Implements the Configuration interface and contains the configuration dictionary for a Managed Service or one of the configuration dictionaries for a Managed Service Factory. These objects are manipulated by configuring bundles.
- *Configuration Plugin* Services – Configuration Plugin services are called before the configuration dictionary is given to the configuration targets. The plug-in can modify the configuration dictionary, which is passed to the Configuration Target.

Figure 9 Configuration Admin Class Diagram org.osgi.service.cm



3.2 Configuration Targets

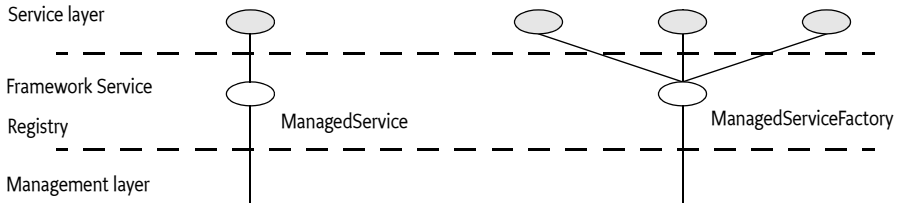
One of the more complicated aspects of this specification is the subtle distinction between the ManagedService and ManagedServiceFactory classes.

Both receive configuration information from the Configuration Admin service and are treated similarly in most respects. Therefore, this specification refers to *configuration targets* when the distinction is irrelevant.

The difference between these types is related to the cardinality of the configuration dictionary. A Managed Service is used when an existing entity needs a configuration dictionary. Thus, a one-to-one relationship always exists between the configuration dictionary and the entity.

A Managed Service Factory is used when part of the configuration is to define *how many instances are required*. A management bundle can create, modify, and delete any number of instances for a Managed Service Factory through the Configuration Admin service. Each instance is configured by a single Configuration object. Therefore, a Managed Service Factory can have multiple associated Configuration objects.

Figure 10 *Differentiation of ManagedService and ManagedServiceFactory Classes*



To summarize:

- A *Managed Service* must receive a single configuration dictionary when it is registered or when its configuration is modified.
- A *Managed Service Factory* must receive from zero to *n* configuration dictionaries when it registers, depending on the current configuration. The Managed Service Factory is informed of configuration dictionary changes: modifications, creations, and deletions.

3.3 The Persistent Identity

A crucial concept in the Configuration Admin service specification is the Persistent IDentity (PID). Its purpose is to act as a primary key for objects that need a configuration dictionary. The name of the service property for PID is defined in the Framework in org.osgi.framework.Constants.SERVICE_PID.

A PID is a unique identifier for a service that persists over multiple invocations of the Framework.

When a bundle registers a service with a PID, it should set property service.pid to a unique value. For that service, the same PID should always be used. If the bundle is stopped and later started, the same PID should be used.

PIDs can be useful for all services, but the Configuration Admin service requires their use with Managed Service and Managed Service Factory registrations because it associates its configuration data with PIDs.

PIDs must be unique for each service. A bundle must not register multiple configuration target services with the same PID. If that should occur, the Configuration Admin service must:

- Send the appropriate configuration data to all services registered under that PID from that bundle only.
- Report an error in the log.
- Ignore duplicate PIDs from other bundles and report them to the log.

3.3.1 PID Syntax

PIDs are intended for use by other bundles, not by people, but sometimes the user is confronted with a PID. For example, when installing an alarm system, the user needs to identify the different components to a wiring application. This type of application exposes the PID to end users.

The schemes for PIDs that are defined in this specification should be followed.

Any globally unique string can be used as a PID. The following sections, however, define schemes for common cases. These schemes are not required, but bundle developers are urged to use them to achieve consistency.

3.3.1.1 Local Bundle PIDs

As a convention, descriptions starting with the bundle identity and a dot (.) are reserved for a bundle. As an example, a PID of "65.536" would belong to the bundle with a bundle identity of 65.

3.3.1.2 Software PIDs

Configuration target services that are singletons can use a Java package name they own as the PID (the reverse domain name scheme). As an example, the PID named com.acme.watchdog would represent a Watchdog service from the ACME company.

3.3.1.3 Devices

Devices are usually organized on buses or networks. The identity of a device, such as a unique serial number or an address, is a good component of a PID. The format of the serial number should be the same as that printed on the housing or box, to aid in recognition..

Bus	Example	Format	Description
USB	USB-0123-0002-9909873	idVendor (hex 4) idProduct (hex 4) iSerialNumber (decimal)	Universal Serial Bus. Use the standard device descriptor.
IP	IP-172.16.28.21	IP nr (dotted decimal)	Internet Protocol
802	802-00:60:97:00:9A:56	MAC address with : separators	IEEE 802 MAC address (Token Ring, Ethernet,...)
ONE	ONE-06-00000021E461	Family (hex 2) and serial number including CRC (hex 6)	1-wire bus of Dallas Semiconductor
COM	COM-krups-brewer-12323	serial number or type name of device	Serial ports

Table 6 *Schemes for Device-Oriented PID Names*

3.4 The Configuration Object

A Configuration object contains the configuration dictionary, which is a set of properties that configure an aspect of a bundle. A bundle can receive Configuration objects by registering a configuration target service with a PID service property. See *The Persistent Identity* on page 27 for more information about PIDs.

During registration, the Configuration Admin service must detect these configuration target services and hand over their configuration dictionary via a callback. If this configuration dictionary is subsequently modified, the modified dictionary is handed over to the configuration target again with the same callback.

The Configuration object is primarily a set of properties that can be updated by a Management Agent, user interfaces on the OSGi Service Platform, or other applications. Configuration changes are first made persistent, and then passed to the target service via a call to the updated method in the ManagedServiceFactory or ManagedService class.

A Configuration object must be uniquely bound to a Managed Service or Managed Service Factory. This implies that a bundle must not register a Managed Service Factory with a PID that is the same as the PID given to a Managed Service.

3.4.1 Location Binding

When a Configuration object is created by either getConfiguration or createFactoryConfiguration, it becomes bound to the location of the calling bundle. This location is obtained with the associated bundle's getLocation method.

Location binding is a security feature that assures that only management bundles can modify configuration data, and other bundles can only modify their own configuration data. A SecurityException is thrown if a bundle other than a Management Agent bundle attempts to modify the configuration information of another bundle.

If a Managed Service is registered with a PID that is already bound to another location, the normal callback to ManagedService.updated must not take place.

The two argument versions of getConfiguration and createFactoryConfiguration take a location String as their second argument. These methods require AdminPermission, and they create Configuration objects bound to the specified location, instead of the location of the calling bundle. These methods are intended for management bundles.

The creation of a Configuration object does not in itself initiate a callback to the target.

A null location parameter may be used to create Configuration objects that are not bound. In this case, the objects become bound to a specific location the first time that they are used by a bundle. When this dynamically bound bundle is subsequently uninstalled, the Configuration object's bundle location must be set to null again so it can be bound again later.

A management bundle may create a Configuration object before the associated Managed Service is registered. It may use a null location to avoid any dependency on the actual location of the bundle which registers this service. When the Managed Service is registered later, the Configuration object must be bound to the location of the registering bundle, and its configuration dictionary must then be passed to ManagedService.updated.

3.4.2 Configuration Properties

A configuration dictionary contains a set of properties in a Dictionary object. The value of the property may be of the following types:

```
type        ::=
   String     | Integer    | Long
 | Float      | Double     | Byte
 | Short      | Character  | Boolean
 | vector     | arrays

primitive   ::= long | int | short | char | byte | double
 | float | boolean

arrays      ::= primitive '[]' | type '[]' | null

vector      ::= Vector of type or null
```

This explicitly allows vectors and arrays of mixed types and containing null.

The name or key of a property must always be a String object, and is not case sensitive during look up, but must preserve the original case.

Bundles should not use nested vectors or arrays, nor should they use mixed types. Using mixed types or nesting makes it impossible to use the meta typing specification. See *Metatype Specification* on page 65.

3.4.3 Property Propagation

An implementation of a Managed Service should copy all the properties of the Dictionary object argument in updated(Dictionary), known or unknown, into its service registration properties using ServiceRegistration.setProperties.

This propagation allows the development of applications that leverage the Framework service registry more extensively, so compliance with this mechanism is advised.

A configuration target service may ignore any configuration properties it does not recognize, or it may change the values of the configuration properties before these properties are registered. Configuration properties in the Framework service registry are not strictly related to the configuration information.

Bundles that cooperate with the propagation of configuration properties can participate in horizontal applications. For example, an application that maintains physical location information in the Framework service registry could find out where a particular device is located in the house or car. This service could use a property dedicated to the physical location and provide functions that leverage this property, such as a graphic user interface that displays these locations.

3.4.4 Automatic Properties

The Configuration Admin service must automatically add a number of properties to the configuration dictionary. If these properties are also set by a configuring bundle or a plug-in, they must always be overridden before they are given to the target service. See *Configuration Plugin* on page 42, Therefore, the receiving bundle or plug-in can assume that the following properties are defined by the Configuration Admin service and not by the configuring bundle:

- service.pid – Set to the PID of the associated Configuration object.
- service.factoryPid – Only set for a Managed Service Factory. It is then set to the PID of the associated Managed Service Factory.
- service.bundleLocation – Set to the location of the bundle that can use this Configuration object. This property can only be used for searching, it may not appear in the configuration dictionary returned from the getProperties method due to security reasons, nor may it be used when the target is updated.

Constants for some of these properties can be found in org.osgi.framework.Constants. These system properties are all of type String.

3.4.5 Equality

Two different Configuration objects can actually represent the same underlying configuration. This means that a Configuration object must implement the equals and hashCode methods in such a way that two Configuration objects are equal when their PID is equal.

3.5 Managed Service

A Managed Service is used by a bundle that needs one configuration dictionary and is thus associated with one Configuration object in the Configuration Admin service.

A bundle can register any number of ManagedService objects, but each must be identified with its own PID.

A bundle should use a Managed Service when it needs configuration information for the following:

- *A Singleton* – A single entity in the bundle that needs to be configured.
- *Externally Detected Devices* – Each device that is detected causes a registration of an associated ManagedService object. The PID of this object is related to the identity of the device, such as the address or serial number.

3.5.1 Networks

When a device in the external world needs to be represented in the OSGi Environment, it must be detected in some manner. The Configuration Admin service cannot know the identity and the number of instances of the device without assistance. When a device is detected, it still needs configuration information in order to play a useful role.

For example, a 1-Wire network can automatically detect devices that are attached and removed. When it detects a temperature sensor, it could register a Sensor service with the Framework service registry. This Sensor service needs configuration information specifically for that sensor, such as which lamps should be turned on, at what temperature the sensor is triggered, what timer should be started, in what zone it resides, and so on. One bundle could potentially have hundreds of these sensors and actuators, and each needs its own configuration information.

Each of these Sensor services should be registered as a Managed Service with a PID related to the physical sensor (such as the address) to receive configuration information.

Other examples are services discovered on networks with protocols like Jini, UPnP, and Salutation. They can usually be represented in the Framework service registry. A network printer, for example, could be detected via UPnP. Once in the service registry, these services usually require local configuration information. A Printer service needs to be configured for its local role: location, access list, and so on.

This information needs to be available in the Framework service registry whenever that particular Printer service is registered. Therefore, the Configuration Admin service must remember the configuration information for this Printer service.

This type of service should register with the Framework as a Managed Service in order to receive appropriate configuration information.

3.5.2 Singletons

When an object must be instantiated only once, it is called a *singleton*. A singleton requires a single configuration dictionary. Bundles may implement several different types of singletons if necessary.

For example, a Watchdog service could watch the registry for the status and presence of services in the Framework service registry. Only one instance of a Watchdog service is needed, so only a single configuration dictionary is required that contains the polling time and the list of services to watch.

3.5.3 Configuring Managed Services

A bundle that needs configuration information should register one or more `ManagedService` objects with a PID service property. If it has a default set of properties for its configuration, it may include them as service properties of the Managed Service. These properties may be used as a configuration template when a `Configuration` object is created for the first time. A Managed Service optionally implements the `MetaTypeProvider` interface to provide information about the property types. See *Meta Typing* on page 46.

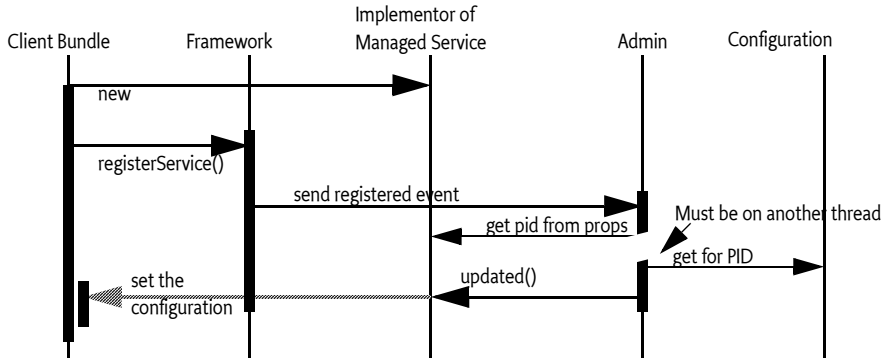
When this registration is detected by the Configuration Admin service, the following steps must occur:

- The configuration stored for the registered PID must be retrieved. If there is a Configuration object for this PID, it is sent to the Managed Service with updated(Dictionary).
- If a Managed Service is registered and no configuration information is available, the Configuration Admin service must call updated(Dictionary) with a null parameter.
- If the Configuration Admin service starts *after* a Managed Service is registered, it must call updated(Dictionary) on this service as soon as possible. For this reason, a Managed Service must always get a callback when it registers *and* the Configuration Admin service is started.

The updated(Dictionary) callback from the Configuration Admin service to the Managed Service must take place asynchronously. This requirement allows the Managed Service to finish its initialization in a synchronized method without interference from the Configuration Admin service callback.

Care should be taken not to cause deadlocks by calling the Framework within a synchronized method.

Figure 11 Managed Service Configuration Action Diagram



The updated method may throw a ConfigurationException. This object must describe the problem and what property caused the exception.

3.5.4 Race Conditions

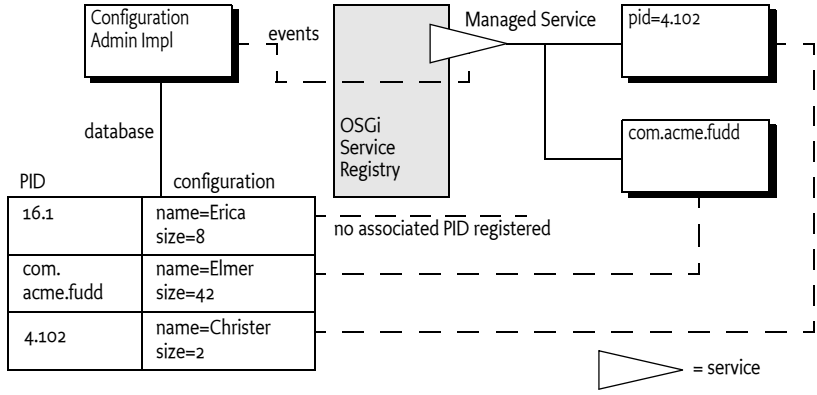
When a Managed Service is registered, the default properties may be visible in the service registry for a short period before they are replaced by the properties of the actual configuration dictionary. Care should be taken that this visibility does not cause race conditions for other bundles.

In cases where race conditions could be harmful, the Managed Service must be split into two pieces: an object performing the actual service and a Managed Service. First, the Managed Service is registered, the configuration is received, and the actual service object is registered. In such cases, the use of a Managed Service Factory that performs this function should be considered.

3.5.5 Examples of Managed Service

Figure 12 shows a Managed Service configuration example. Two services are registered under the ManagedService interface, each with a different PID.

Figure 12 *PIDs and External Associations*



The Configuration Admin service has a database containing a configuration record for each PID. When the Managed Service with service.pid = com.acme.fudd is registered, the Configuration Admin service will retrieve the properties name=Elmer and size=42 from its database. The properties are stored in a Dictionary object and then given to the Managed Service with the updated(Dictionary) method.

3.5.5.1 Configuring A Console Bundle

In this example, a bundle can run a single debugging console over a Telnet connection. It is a singleton, so it uses a ManagedService object to get its configuration information: the port and the network name on which it should register.

```
class SampleManagedService implements ManagedService {
   Dictionary              properties;
   ServiceRegistration     registration;
   Console                 console;

   public synchronized void start(
      BundleContext context ) throws Exception {
      properties = new Hashtable();
      properties.put( Constants.SERVICE_PID,
         "com.acme.console" );
      properties.put( "port",  new Integer(2011) );

      registration = context.registerService(
         ManagedService.class.getName(),
         this,
         properties
      );
   }
```

```
    public synchronized void updated( Dictionary np ) {
       if ( np != null ) {
          properties = np;
          properties.put(
             Constants.SERVICE_PID, "com.acme.console" );
       }

       if (console == null)
          console = new Console();

       int port = ((Integer)properties.get("port"))
          .intValue();

       String network = (String) properties.get("network");
       console.setPort(port, network);
       registration.setProperties(properties);
    }
    ... further methods
}
```

3.5.6 Deletion

When a Configuration object for a Managed Service is deleted, the Configuration Admin service must call updated(Dictionary) with a null argument on a thread that is different from that on which the Configuration.delete was executed.

3.6 Managed Service Factory

A Managed Service Factory is used when configuration information is needed for a service that can be instantiated multiple times. When a Managed Service Factory is registered with the Framework, the Configuration Admin service consults its database and calls updated(String,Dictionary) for each associated Configuration object. It passes the identifier of the instance, which can be used as a PID, as well as a Dictionary object with the configuration properties.

A Managed Service Factory is useful when the bundle can provide functionality a number of times, each time with different configuration dictionaries. In this situation, the Managed Service Factory acts like a *class* and the Configuration Admin service can use this Managed Service Factory to *instantiate instances* for that *class*.

In the next section, the word *factory* refers to this concept of creating *instances* of a function defined by a bundle that registers a Managed Service Factory.

3.6.1 When to Use a Managed Service Factory

A Managed Service Factory should be used when a bundle does not have an internal or external entity associated with the configuration information but can potentially be instantiated multiple times.

3.6.1.1 Example Email Fetcher

An email fetcher program displays the number of emails that a user has – a function likely to be required for different users. This function could be viewed as a *class* that needs to be *instantiated* for each user. Each instance requires different parameters, including password, host, protocol, user id, and so on.

An implementation of the Email Fetcher service should register a ManagedServiceFactory object. In this way, the Configuration Admin service can define the configuration information for each user separately. The Email Fetcher service will only receive a configuration dictionary for each required instance (user).

3.6.1.2 Example Temperature Conversion Service

Assume a bundle has the code to implement a conversion service that receives a temperature and, depending on settings, can turn an actuator on and off. This service would need to be instantiated many times depending on where it is needed. Each instance would require its own configuration information for the following:

- Upper value
- Lower value
- Switch Identification
- ...

Such a conversion service should register a service object under a ManagedServiceFactory interface. A configuration program can then use this Managed Service Factory to create instances as needed. For example, this program could use a Graphic User Interface (GUI) to create such a component and configure it.

3.6.1.3 Serial Ports

Serial ports cannot always be used by the OSGi Device Access specification implementations. Some environments have no means to identify available serial ports, and a device on a serial port cannot always provide information about its type.

Therefore, each serial port requires a description of the device that is connected. The bundle managing the serial ports would need to instantiate a number of serial ports under the control of the Configuration Admin service, with the appropriate DEVICE_CATEGORY property to allow it to participate in the Device Access implementation.

If the bundle cannot detect the available serial ports automatically, it should register a Managed Service Factory. The Configuration Admin service can then, with the help of a configuration program, define configuration information for each available serial port.

3.6.2 Registration

Similar to the Managed Service configuration dictionary, the configuration dictionary for a Managed Service Factory is identified by a PID. The Managed Service Factory, however, also has a *factory* PID, which is the PID of the associated Managed Service Factory. It is used to group all Managed Service Factory configuration dictionaries together.

When a Configuration object for a Managed Service Factory is created (ConfigurationAdmin.createFactoryConfiguration), a new unique PID is created for this object by the Configuration Admin service. The scheme used for this PID is defined by the Configuration Admin service and is unrelated to the factory PID.

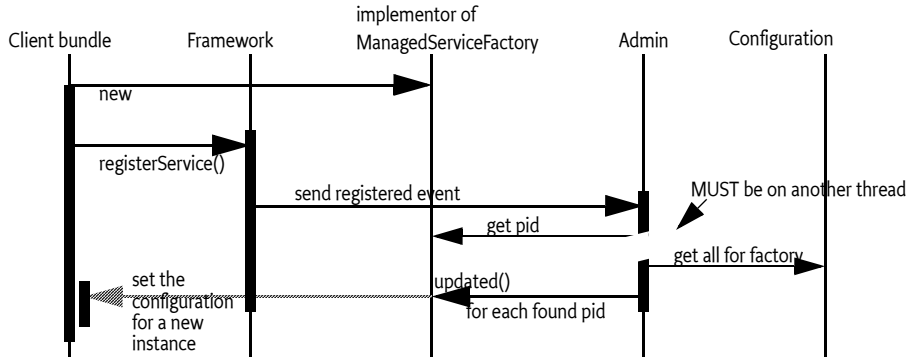
When the Configuration Admin service detects the registration of a Managed Service Factory, it must find all configuration dictionaries for this factory and must then sequentially call ManagedServiceFactory.updated(String,Dictionary) for each configuration dictionary. The first argument is the PID of the Configuration object (the one created by the Configuration Admin service) and the second argument contains the configuration properties.

The Managed Service Factory should then create instances of the associated factory class. Using the PID given in the Configuration object, the bundle may register new services (other than a Managed Service) with the Framework, but this is not required. This may be necessary when the PID is useful in contexts other than the Configuration Admin service.

The receiver must *not* register a Managed Service with this PID because this would force two Configuration objects to have the same PID. If a bundle attempts to do this, the Configuration Admin service should log an error and must ignore the registration of the Managed Service. The configuration dictionary may be used only internally.

The Configuration Admin service must guarantee that the Configuration objects are not deleted before their properties are given to the Managed Service Factory, and must assure that no race conditions exist between initialization and updates.

Figure 13 *Managed Service Factory Action Diagram*



A Managed Service Factory has only one update method: updated(String, Dictionary). This method can be called any number of times as Configuration objects are created or updated.

The Managed Service Factory must detect whether a PID is being used for the first time, in which case it should create a new *instance*, or a subsequent time, in which case it should update an existing instance.

The Configuration Admin service must call updated(String,Dictionary) on a thread that is different from the one that executed the registration. This requirement allows an implementation of a Managed Service Factory to use a synchronized method to assure that the callbacks do not interfere with the Managed Service Factory registration.

The updated(String,Dictionary) method may throw a ConfigurationException object. This object describes the problem and what property caused the problem. These exceptions should be logged by a Configuration Admin service.

3.6.3 Deletion

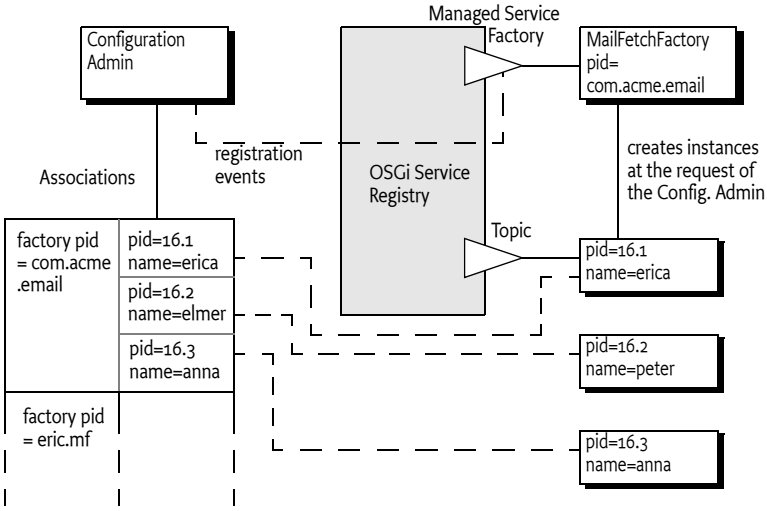
If a configuring bundle deletes an instance of a Managed Service Factory, the deleted(String) method is called. The argument is the PID for this instance. The implementation of the Managed Service Factory must remove all information and stop any behavior associated with that PID. If a service was registered for this PID, it should be unregistered.

3.6.4 Managed Service Factory Example

Figure 14 highlights the differences between a Managed Service and a Managed Service Factory. It shows how a Managed Service Factory implementation receives configuration information that was created before it was registered.

- A bundle implements an EMail Fetcher service. It registers a ManagedServiceFactory object with PID=com.acme.email.
- The Configuration Admin service notices the registration and consults its database. It finds three Configuration objects for which the factory PID is equal to com.acme.email. It must call updated(String,Dictionary) for each of these Configuration objects on the newly registered ManagedServiceFactory object.
- For each configuration dictionary received, the factory should create a new instance of a EMailFetcher object, one for erica (PID=16.1), one for anna (PID=16.3), and one for elmer (PID=16.2).
- The EMailFetcher objects are registered under the Topic interface so their results can be viewed by an online display.
  If the EMailFetcher object is registered, it may safely use the PID of the Configuration object because the Configuration Admin service must guarantee its suitability for this purpose.

Figure 14 *Managed Service Factory Example*



3.6.5 Multiple Consoles Example

This example illustrates how multiple consoles, each of which has its own port and interface can run simultaneously. This approach is very similar to the example for the Managed Service, but highlights the difference by allowing multiple consoles to be created.

```
class ExampleFactory implements ManagedServiceFactory {
   Hashtable consoles = new Hashtable();
   BundleContext context;
   public void start( BundleContext context )
      throws Exception {
      this.context = context;
      Hashtable local = new Hashtable();
      local.put(Constants.SERVICE_PID,"com.acme.console");
      context.registerService(
         ManagedServiceFactory.class.getName(),
         this,
         local );
   }

   public void updated( String pid, Dictionary config ){
      Console console = (Console) consoles.get(pid);
      if (console == null) {
         console = new Console(context);
         consoles.put(pid, console);
      }

      int port = getInt(config, "port", 2011);
      String network = getString(
         config,
         "network",
         null /*all*/
```

```
            );
            console.setPort(port, network);
        }

        public void deleted(String pid) {
            Console console = (Console) consoles.get(pid);
            if (console != null) {
                consoles.remove(pid);
                console.close();
            }
        }
    }
```

3.7 Configuration Admin Service

The ConfigurationAdmin interface provides methods to maintain configuration data in an OSGi environment. This configuration information is defined by a number of Configuration objects associated with specific configuration targets. Configuration objects can be created, listed, modified, and deleted through this interface. Either a remote management system or the bundles configuring their own configuration information may perform these operations.

The ConfigurationAdmin interface has methods for creating and accessing Configuration objects for a Managed Service, as well as methods for managing new Configuration objects for a Managed Service Factory.

3.7.1 Creating a Managed Service Configuration Object

A bundle can create a new Managed Service Configuration object with ConfigurationAdmin.getConfiguration. No create method is offered because doing so could introduce race conditions between different bundles creating the same Configuration object. The getConfiguration method must atomically create and persistently store an object if it does not yet exist.

Two variants of this method are:

- getConfiguration(String) – This method is used by a bundle with a given location to configure its *own* ManagedService objects. The argument specifies the PID of the targeted service.
- getConfiguration(String,String) – This method is used by a management bundle to configure *another* bundle. Therefore, this management bundle needs AdminPermission. The first argument is the PID and the second argument is the location identifier of the targeted ManagedService object.

All Configuration objects have a method, getFactoryPid(), which in this case must return null because the Configuration object is associated with a Managed Service.

Creating a new Configuration object must *not* initiate a callback to the Managed Service updated method.

3.7.2 Creating a Managed Service Factory Configuration Object

The ConfigurationAdmin class provides two methods to create a new instance of a Managed Service Factory:

- createFactoryConfiguration(String) – This method is used by a bundle with a given location to configure its own ManagedServiceFactory objects. The argument specifies the PID of the targeted ManagedServiceFactory object. This *factory PID* can be obtained from the returned Configuration object with the getFactoryPid() method.
- createFactoryConfiguration(String,String)– This method is used by a management bundle to configure another bundle's ManagedServiceFactory object. This management bundle needs AdminPermission. The first argument is the location identifier and the second is the PID of the targeted ManagedServiceFactory object. The *factory PID* can be obtained from the returned Configuration object with getFactoryPid method.

Creating a new factory configuration must *not* initiate a callback to the Managed Service Factory updated method until the properties are set in the Configuration object.

3.7.3 Accessing Existing Configurations

The existing set of Configuration objects can be listed with listConfigurations(String). The argument is a String object with a filter expression. This filter expression has the same syntax as the Framework Filter class. For example:

```
(&(size=42)(service.factoryPid=*osgi*))
```

The filter function must use the properties of the Configuration objects and only return the ones that match the filter expression.

A single Configuration object is identified with a PID and can be obtained with getConfiguration(String).

If the caller has AdminPermission, then all Configuration objects are eligible for search. In other cases, only Configuration objects bound to the calling bundle's location must be returned.

null is returned in both cases when an appropriate Configuration object cannot be found.

3.7.3.1 Updating a Configuration

The process of updating a Configuration object is the same for Managed Services and Managed Service Factories. First, listConfigurations(String) or getConfiguration(String) should be used to get a Configuration object. The properties can be obtained with Configuration.getProperties. When no update has occurred since this object was created, getProperties returns null.

New properties can be set by calling Configuration.update. The Configuration Admin service must first store the configuration information and then call a configuration target's updated method: either the ManagedService.updated or ManagedServiceFactory.updated method. If this target service is not registered, the fresh configuration information must be set when the configuration target service registers.

The update method calls in Configuration objects are not executed synchronously with the related target service updated method. This method must be called asynchronously. The Configuration Admin service, however, must have updated the persistent storage before the update method returns.

3.7.4 Deletion

A Configuration object that is no longer needed can be deleted with Configuration.delete, which removes the Configuration object from the database. The database must be updated before the target service updated method is called.

If the target service is a Managed Service Factory, the factory is informed of the deleted Configuration object by a call to ManagedServiceFactory.deleted. It should then remove the associated *instance*. The ManagedServiceFactory.deleted call must be done asynchronously with respect to Configuration.delete.

When a Configuration object of a Managed Service is deleted, ManagedService.updated is called with null for the properties argument. This method may be used for clean-up, to revert to default values, or to unregister a service.

3.7.5 Updating a Bundle's Own Configuration

The Configuration Admin service specification does not distinguish between updates via a Management Agent and a bundle updating its own configuration information (as defined by its location). Even if a bundle updates its own configuration information, the Configuration Admin service must callback the associated target service updated method.

As a rule, to update its own configuration, a bundle's user interface should *only* update the configuration information and never its internal structures directly. This rule has the advantage that the events, from the bundle implementation's perspective, appear similar for internal updates, remote management updates, and initialization.

3.8 Configuration Plugin

The Configuration Admin service allows third-party applications to participate in the configuration process. Bundles that register a service object under a ConfigurationPlugin interface can process the configuration dictionary just before it reaches the configuration target service.

Plug-ins allow sufficiently privileged bundles to intercept configuration dictionaries just *before* they must be passed to the intended Managed Service or Managed Service Factory but *after* the properties are stored. The changes the plug-in makes are dynamic and must not be stored. The plug-in must only be called when an update takes place while it is registered.

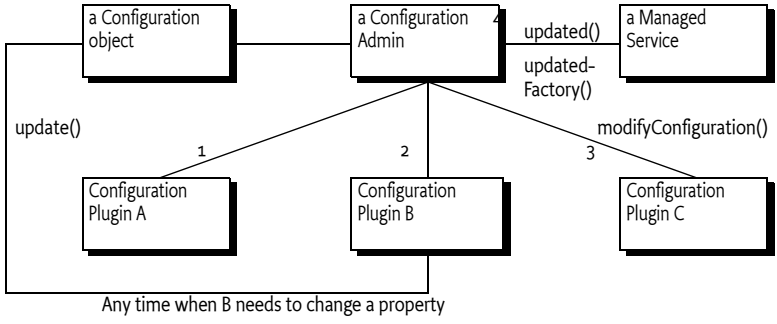
The ConfigurationPlugin interface has only one method: modifyConfiguration(ServiceReference,Dictionary). This method inspects or modifies configuration data.

All plug-ins in the service registry must be traversed and called before the properties are passed to the configuration target service. Each Configuration Plugin object gets a chance to inspect the existing data, look at the target object, which can be a ManagedService object or a ManagedServiceFactory object, and modify the properties of the configuration dictionary. The changes made by a plug-in must be visible to plugins that are called later.

ConfigurationPlugin objects should not modify properties that belong to the configuration properties of the target service unless the implications are understood. This functionality is mainly intended to provide functions that leverage the Framework service registry. The changes made by the plugin should normally not be validated. However, the Configuration Admin must ignore changes to the automatic properties as described in *Automatic Properties* on page 31.

For example, a Configuration Plugin service may add a physical location property to a service. This property can be leveraged by applications that want to know where a service is physically located. This scenario could be carried out without any further support of the service itself, except for the general requirement that the service should propagate the properties it receives from the Configuration Admin service to the service registry.

Figure 15 *Order of Configuration Plugin Services*



3.8.1 Limiting The Targets

A ConfigurationPlugin object may optionally specify a cm.target registration property. This value is the PID of the configuration target whose configuration updates the ConfigurationPlugin object wants to intercept.

The ConfigurationPlugin object must then only be called with updates for the configuration target service with the specified PID. Omitting the cm.target registration property means that it is called for *all* configuration updates.

3.8.2 Example of Property Expansion

Consider a Managed Service that has a configuration property service.to with the value (objectclass=com.acme.Alarm). When the Configuration Admin service sets this property on the target service, a ConfigurationPlugin object may replace the (objectclass=com.acme.Alarm) filter with an array of existing alarm systems' PIDs as follows:

```
ID "service.to=[32434,232,12421,1212]"
```

A new Alarm Service with service.pid=343 is registered, requiring that the list of the target service be updated. The bundle which registered the Configuration Plugin service, therefore, wants to set the to registration property on the target service. It does *not* do this by calling ManagedService.updated directly for several reasons:

- In a securely configured system, it should not have the permission to make this call or even obtain the target service.
- It could get into race conditions with the Configuration Admin service if it had the permissions in the previous bullet. Both services would compete for access simultaneously.

Instead, it must get the Configuration object from the Configuration Admin service and call the update method on it.

The Configuration Admin service must schedule a new update cycle on another thread, and sometime in the future must call ConfigurationPlugin.modifyProperties. The ConfigurationPlugin object could then set the service.to property to [32434,232,12421,1212, 343]. After that, the Configuration Admin service must call updated on the target service with the new service.to list.

3.8.3 Configuration Data Modifications

Modifications to the configuration dictionary are still under the control of the Configuration Admin service, which must determine whether to accept the changes, hide critical variables, or deny the changes for other reasons.

The ConfigurationPlugin interface must also allow plugins to detect configuration updates to the service via the callback. This ability allows them to synchronize the configuration updates with transient information.

3.8.4 Forcing a Callback

If a bundle needs to force a Configuration Plugin service to be called again, it must fetch the appropriate Configuration object from the Configuration Admin service and call the update() method (the no parameter version) on this object. This call forces an update with the current configuration dictionary so that all applicable plug-ins get called again.

3.8.5 Calling Order

The order in which the ConfigurationPlugin objects are called must depend on the service.cmRanking configuration property of the ConfigurationPlugin object. Table 7 shows the usage of the service.cmRanking property for the order of calling the Configuration Plugin services..

service.cmRanking value	Description
< 0	The Configuration Plugin service should not modify properties and must be called before any modifications are made.
> 0 && <= 1000	The Configuration Plugin service modifies the configuration data. The calling order should be based on the value of the service.cmRanking property.
> 1000	The Configuration Plugin service should not modify data and is called after all modifications are made.

Table 7 service.cmRanking *Usage For Ordering*

3.9 Remote Management

This specification does not attempt to define a remote management interface for the Framework. The purpose of this specification is to define a minimal interface for bundles that is complete enough for testing.

The Configuration Admin service is a primary aspect of remote management, however, and this specification must be compatible with common remote management standards. This section discusses some of the issues of using this specification with [4] *DMTF Common Information Model* (CIM) and [5] *Simple Network Management Protocol* (SNMP), the most likely candidates for remote management today.

These discussions are not complete, comprehensive, or normative. They are intended to point the bundle developer in relevant directions. Further specifications are needed to make a more concrete mapping.

3.9.1 Common Information Model

Common Information Model (CIM) defines the managed objects in [7] *Interface Definition Language* (IDL) language, which was developed for the Common Object Request Broker Architecture (CORBA).

The data types and the data values have a syntax. Additionally, these syntaxes can be mapped to XML. Unfortunately, this XML mapping is very different from the very applicable [6] *XSchema* XML data type definition language. The Framework service registry property types are a proper subset of the CIM data types.

In this specification, a Managed Service Factory maps to a CIM class definition. The primitives create, delete, and set are supported in this specification via the ManagedServiceFactory interface. The possible data types in CIM are richer than those the Framework supports and should thus be limited to cases when CIM classes for bundles are defined.

An important conceptual difference between this specification and CIM is the naming of properties. CIM properties are defined within the scope of a class. In this specification, properties are primarily defined within the scope of the Managed Service Factory, but are then placed in the registry, where they have global scope. This mechanism is similar to [8] *Lightweight Directory Access Protocol*, in which the semantics of the properties are defined globally and a class is a collection of globally defined properties.

This specification does not address the non-Configuration Admin service primitives such as notifications and method calls.

3.9.2 Simple Network Management Protocol

The Simple Network Management Protocol (SNMP) defines the data model in ASN.1. SNMP is a rich data typing language that supports many types that are difficult to map to the data types supported in this specification. A large overlap exists, however, and it should be possible to design a data type that is applicable in this context.

The PID of a Managed Service should map to the SNMP Object IDentifier (OID). Managed Service Factories are mapped to tables in SNMP, although this mapping creates an obvious restriction in data types because tables can only contain scalar values. Therefore, the property values of the Configuration object would have to be limited to scalar values.

Similar scope issues as seen in CIM arise for SNMP because properties have a global scope in the service registry.

SNMP does not support the concept of method calls or function calls. All information is conveyed as the setting of values. The SNMP paradigm maps closely to this specification.

This specification does not address non-Configuration Admin primitives such as traps.

3.10 Meta Typing

This section discusses how the Metatype specification is used in the context of a Configuration Admin service.

When a Managed Service or Managed Service Factory is registered, the service object may also implement the MetaTypeProvider interface.

If the Managed Service or Managed Service Factory object implements the MetaTypeProvider interface, a management bundle may assume that the associated ObjectClassDefinition object can be used to configure the service.

The ObjectClassDefinition and AttributeDefinition objects contain sufficient information to automatically build simple user interfaces. They can also be used to augment dedicated interfaces with accurate validations.

When the Metatype specification is used, care should be taken to match the capabilities of the metatype package to the capabilities of the Configuration Admin service specification. Specifically:

- The metatype specification must describe nested arrays and vectors or arrays/vectors of mixed type.

This specification does not address how the metatype is made available to a management system due to the many open issues regarding remote management.

3.11 Security

3.11.1 Permissions

Configuration Admin service security is implemented using ServicePermission and AdminPermission. The following table summarizes the permissions needed by the Configuration Admin bundle itself, as well as those needed by the bundles with which it interacts.

Bundle Registering	ServicePermisson Action	AdminPermission
ConfigurationAdmin	REGISTER ConfigurationAdmin	Yes
	GET ManagedService	
	GET ManagedServiceFactory	
	GET ConfigurationPlugin	
ManagedService	REGISTER ManagedService	No
	GET ConfigurationAdmin	
ManagedServiceFactory	REGISTER ManagedServiceFactory	No
	GET ConfigurationAdmin	
ConfigurationPlugin	REGISTER ConfigurationPlugin	No
	GET ConfigurationAdmin	

Table 8 *Permission Overview Configuration Admin*

The Configuration Admin service must have ServicePermission[REGISTER, ConfigurationAdmin]. It will also be the only bundle that needs the ServicePermission[GET,ManagedService | ManagedServiceFactory |ConfigurationPlugin]. No other bundle should be allowed to have GET permission for these interfaces. The Configuration Admin bundle must also hold AdminPermission.

Bundles that can be configured must have the ServicePermission[REGISTER,ManagedService |ManagedServiceFactory].

Bundles registering ConfigurationPlugin objects must have the ServicePermission[REGISTER, ConfigurationPlugin]. The Configuration Admin service must trust all services registered with the ConfigurationPlugin interface. Only the Configuration Admin service should have ServicePermission[GET, ConfigurationPlugin.

If a Managed Service or Managed Service Factory is implemented by an object that is also registered under another interface, it is possible, although inappropriate, for a bundle other than the Configuration Admin service implementation to call the updated method. Security-aware bundles can avoid this problem by having their updated methods check that the caller has AdminPermission (such bundles need AdminPermission to perform this check).

Bundles that want to change their own configuration need ServicePermission[GET, ConfigurationAdmin]. A bundle with AdminPermission is allowed to access and modify any Configuration object.

Pre-configuration of bundles requires AdminPermission because the methods that specify a location require this permission.

3.11.2 Forging PIDs

A risk exists of an unauthorized bundle forging a PID in order to obtain and possibly modify the configuration information of another bundle. To mitigate this risk, Configuration objects are generally *bound* to a specific bundle location, and are not passed to any Managed Service or Managed Service Factory registered by a different bundle.

Bundles with the required AdminPermission can create Configuration objects that are not bound. In other words, they have their location set to null. This can be useful for preconfiguring bundles before they are installed without having to know their actual locations.

In this scenario, the Configuration object must become bound to the first bundle that registers a Managed Service (or Managed Service Factory) with the right PID.

A bundle could still possibly obtain another bundle's configuration by registering a Managed Service with the right PID before the victim bundle does so. This situation can be regarded as a denial-of-service attack, because the victim bundle would never receive its configuration information. Such an attack can be avoided by always binding Configuration objects to the right locations. It can also be detected by the Configuration Admin service when the victim bundle registers the correct PID and two equal PIDs are then registered. This violation of this specification should be logged.

3.11.3 Configuration and Permission Administration

Configuration information has a direct influence on the permissions needed by a bundle. For example, when the Configuration Admin Bundle orders a bundle to use port 2011 for a console, that bundle also needs permission for listening to incoming connections on that port.

Both a simple and a complex solution exist for this situation.

The simple solution for this situation provides the bundle with a set of permissions that do not define specific values but allow a range of values. For example, a bundle could listen to ports above 1024 freely. All these ports could then be used for configuration.

The other solution is more complicated. In an environment where there is very strong security, the bundle would only be allowed access to a specific port. This situation requires an atomic update of both the configuration data and the permissions. If this update was not atomic, a potential security hole would exist during the period of time that the set of permissions did not match the configuration.

The following scenario can be used to update a configuration and the security permissions:

1. Stop the bundle.
2. Update the appropriate Configuration object via the Configuration Admin service.
3. Update the permissions in the Framework.
4. Start the bundle.

This scenario would achieve atomicity from the point of view of the bundle.

3.12 Configurable Service

Both the Configuration Admin service and the org.osgi.framework.Configurable interface address configuration management issues. It is the intention of this specification to replace the Framework interface for configuration management.

The Framework Configurable mechanism works as follows. A registered service object implements the Configurable interface to allow a management bundle to configure that service. The Configurable interface has only one method: getConfigurationObject(). This method returns a Java Bean. Beans can be examined and modified with the java.reflect or java.bean packages.

This scheme has the following disadvantages:

- *No factory* – Only registered services can be configured, unlike the Managed Service Factory that configures any number of services.
- *Atomicity* – The beans or reflection API can only modify one property at a time and there is no way to tell the bean that no more modifications to the properties will follow. This limitation complicates updates of configurations that have dependencies between properties.
  This specification passes a Dictionary object that sets all the configuration properties atomically.
- *Profile* – The Java beans API is linked to many packages that are not likely to be present in OSGi environments. The reflection API may be present but is not simple to use.
  This specification has no required libraries.
- *User Interface support* – UI support in beans is very rudimentary when no AWT is present.
  The associated Metatyping specification does not require any external libraries, and has extensive support for UIs including localization.

3.13 Changes

3.13.1 Clarifications

- It was not clear from the description that a PID received through a Managed Service Factory must not be used to register a Managed Service. This has been highlighted in the appropriate sections.
- It was not clearly specified that a call-back to a target only happens when the data is updated or the target is registered. The creation of a Configuration object does not initiate a call-back. This has been highlighted in the appropriate sections.
- In this release, when a bundle is uninstalled, all Configuration objects that are dynamically bound to that bundle must be unbound again. See *Location Binding* on page 29.
- It was not clearly specified that the data types of a Configuration object allow arrays and vectors that contain elements of mixed types and also null.

3.13.2 Removal of Bundle Location Property

The bundle location property that was required to be set in the Configuration object's properties has been removed because it leaked security sensitive information to all bundles using the Configuration object.

3.13.3 Plug-in Usage

It was not completely clear when a plug-in must be called and how the properties dictionary should behave. This has been clearly specified in *Configuration Plugin* on page 42.

3.13.4 BigInteger/BigDecimal

The classes BigInteger and BigDecimal are not part of the minimal execution requirements and are therefore no longer part of the supported Object types in the Configuration dictionary.

3.13.5 Equals

The behavior of the equals and hashCode methods is now defined. See *Equality* on page 31.

3.13.6 Constant for service.factoryPid

Added a new constant in the ConfigurationAdmin class. See *SERVICE_FACTORYPID* on page 55. This caused this specification to step from version 1.0 to version 1.1.

3.14 org.osgi.service.cm

The OSGi Configuration Admin service Package. Specification Version 1.2

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.cm; specification-version=1.2
```

3.14.1 Summary

- Configuration - The configuration information for a ManagedService or ManagedServiceFactory object. [p.51]
- ConfigurationAdmin - Service for administering configuration data. [p.54]
- ConfigurationException - An Exception class to inform the Configuration Admin service of problems with configuration data. [p.57]
- ConfigurationListener - Listener for Configuration changes. [p.57]
- ConfigurationPlugin - A service interface for processing configuration dictionary before the update. [p.58]
- ManagedService - A service that can receive configuration data from a Configuration Admin service. [p.60]
- ManagedServiceFactory - Manage multiple service instances. [p.62]

3.14.2 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 is preserved from the last set key/value.

A configuration can be *bound* to a bundle location (Bundle.getLocation()). The purpose of binding a Configuration object to a location is to make it impossible for another bundle to forge a PID that would match this configuration. When a configuration is bound to a specific location, and a bundle with a different location registers a corresponding ManagedService object or ManagedServiceFactory object, then the configuration is not passed to the updated method of that object.

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.

3.14.2.1 public 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 intiates a call to any ConfigurationListeners asynchronously.

Throws IOException – If delete fails

IllegalStateException – if this configuration has been deleted

3.14.2.2 public boolean equals(Object other)

other Configuration object to compare against

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

Returns true if equal, false if not a Configuration object or one with a different PID.

3.14.2.3 public String getBundleLocation( )

- Get the bundle location. Returns the bundle location to which this configuration is bound, or null if it is not yet bound to a bundle location.

  This call requires AdminPermission.

Returns location to which this configuration is bound, or null.

Throws SecurityException – if the caller does not have AdminPermission.

IllegalStateException – if this Configuration object has been deleted.

3.14.2.4 public 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

3.14.2.5 public String getPid( )

- Get the PID for this Configuration object.

Returns the PID for this Configuration object.

Throws IllegalStateException – if this configuration has been deleted

3.14.2.6 public 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

3.14.2.7 public int hashCode( )

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

Returns hash code for this Configuration object

3.14.2.8 public void setBundleLocation(String bundleLocation)

bundleLocation a bundle location or null

- Bind this Configuration object to the specified bundle 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.

  This method requires AdminPermission.

Throws SecurityException – if the caller does not have AdminPermission

IllegalStateException – if this configuration has been deleted

3.14.2.9 public void update(Dictionary properties) throws IOException

properties the new set of properties for this configuration

- 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 intiates a call to any ConfigurationListeners asynchronously.

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

3.14.2.10 public 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. Also intiates a call to any ConfigurationListeners asynchronously.

  This is the only way for a bundle that uses a Configuration Plugin service to initate 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[p.58]

3.14.3 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 PID matches the PID registration property (service.pid) of the Managed Service. If found, it calls ManagedService.updated[p.61] 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 factoryPid matches the PID 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 another bundle requires AdminPermission.

Configuration objects can be *bound* to a specified bundle location. In this case, if a matching Managed Service or Managed Service Factory is registered by a bundle with a different location, then the Configuration Admin service must not do the normal callback, and it should log an error. In the case where a Configuration object is not bound, its location field is null, the Configuration Admin service will bind it to the location of the bundle

that registers the first Managed Service or Managed Service Factory that has a corresponding PID property. When a Configuration object is bound to a bundle location in this manner, the Confguration Admin service must detect if the bundle corresponding to the location is uninstalled. If this occurs, the Configuration object is unbound, that is its location field is set back to null.

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.

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

Service 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

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

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

Since 1.1

3.14.3.3 public Configuration createFactoryConfiguration(String factoryPid) throws IOException

factoryPid PID of factory (not null).

- 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)[p.53] 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.

Returns a new Configuration object.

Throws IOException – if access to persistent storage fails.

SecurityException – if caller does not have AdminPermission and factoryPid is bound to another bundle.

3.14.3.4 public Configuration createFactoryConfiguration(String factoryPid, String location) throws IOException

factoryPid PID of factory (not null).

location a bundle location string, or null.

- 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)[p.53] 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.

This method requires AdminPermission.

Returns a new Configuration object.

Throws IOException – if access to persistent storage fails.

SecurityException – if caller does not have AdminPermission.

3.14.3.5 public Configuration getConfiguration(String pid, String location) throws IOException

pid persistent identifier.

location the bundle location string, or null.

□ 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.

This method requires AdminPermission.

Returns an existing or new Configuration object.

Throws IOException – if access to persistent storage fails.

SecurityException – if the caller does not have AdminPermission.

3.14.3.6 public Configuration getConfiguration(String pid) throws IOException

pid persistent identifier.

□ 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.

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

If the location of the Configuration object does not match the calling bundle, throw a SecurityException.

Returns an existing or new Configuration matching the PID.

Throws IOException – if access to persistent storage fails.

SecurityException – if the Configuration object is bound to a location different from that of the calling bundle and it has no AdminPermission.

3.14.3.7 public Configuration[] listConfigurations(String filter) throws

IOException, InvalidSyntaxException

filter a `Filter` object, or `null` to retrieve all `Configuration` objects.

- 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.

  Normally only `Configuration` objects that are bound to the location of the calling bundle are returned. If the caller has `AdminPermission`, then all matching `Configuration` objects are returned.

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

  - `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.

Returns all matching `Configuration` objects, or `null` if there aren't any

Throws `IOException` – if access to persistent storage fails

`InvalidSyntaxException` – if the filter string is invalid

3.14.4 public class ConfigurationException extends Exception

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

3.14.4.1 public ConfigurationException(String property, String reason)

property name of the property that caused the problem, `null` if no specific property was the cause

reason reason for failure

- Create a `ConfigurationException` object.

3.14.4.2 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

3.14.4.3 public String getReason( )

- Return the reason for this exception.

Returns reason of the failure

3.14.5 public interface ConfigurationListener

Listener for Configuration changes.

ConfigurationListener objects are registered with the Framework service registry and are notified when a Configuration object is updated or deleted.

ConfigurationListener objects are passed the type of configuration change.

One of the change methods will be called with CM_UPDATED when Configuration.update is called or with CM_DELETED when Configuration.delete is called. Notification will be asynchronous to the update or delete method call. The design is very lightweight in that is does not pass Configuration objects, the listener is merely advised that the configuration information for a given pid has changed. If the listener wants to locate the Configuration object for the specified pid, it must use ConfigurationAdmin.

Security Considerations. Bundles wishing to monitor Configuration changes will require ServicePermission[ConfigurationListener, REGISTER] to register a ConfigurationListener service. Since Configuration objects are not passed to the listener, no sensitive configuration information is available to the listener.

3.14.5.1 public static final int CM_DELETED = 2

Change type that indicates that Configuration.delete was called.

3.14.5.2 public static final int CM_UPDATED = 1

Change type that indicates that Configuration.update was called.

3.14.5.3 public void configurationChanged(String pid, int type)

pid The pid of the configuration which changed.

type The type of the configuration change.

- Receives notification a configuration has changed.

  This method is only called if the target of the configuration is a ManagedService.

3.14.5.4 public void factoryConfigurationChanged(String factoryPid, String pid, int type)

factoryPid The factory pid for the changed configuration.

pid The pid of the configuration which changed.

type The type of the configuration change.

- Receives notification a factory configuration has changed.

  This method is only called if the target of the configuration is a ManagedServiceFactory.

3.14.6 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 ManagedS ervice or Managed Service Factory.

Configuration Plugin (plugin) services have full read/write access to all configuration information. Therefore, bundles using this facility should be trusted. Access to this facility should be limited with ServicePermission[REGISTER, ConfigurationPlugin]. Implementations of a Configuration Plugin service should assure that they only act on appropriate configurations.

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 targetted 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.

3.14.6.1 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

3.14.6.2 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.

3.14.6.3 public void modifyConfiguration(ServiceReference reference, Dictionary properties)

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.

- 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.

3.14.7 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 identitifier 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,
      null // Properties will come from CM in updated
    );
  }

  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.

3.14.7.1 public void updated(Dictionary properties) throws ConfigurationException

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.

- 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 which 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.

Throws `ConfigurationException` – when the update fails

3.14.8 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 );
    }
    ...
  }
```

3.14.8.1 public void deleted(String pid)

pid the PID of the service to be removed

□ 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.

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.

3.14.8.2 public String getName( )

□ Return a descriptive name of this factory.

Returns the name for the factory, which might be localized

3.14.8.3 public void updated(String pid, Dictionary properties) throws ConfigurationException

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.

- 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`[p.57] 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.

Throws `ConfigurationException` – when the configuration properties are invalid.

3.15 References

[4] *DMTF Common Information Model*
http://www.dmtf.org

[5] *Simple Network Management Protocol*
RFCs http://directory.google.com/Top/Computers/Internet/Protocols/SNMP/RFCs

[6] *XSchema*
http://www.w3.org/TR/xmlschema-0/

[7] *Interface Definition Language*
http://www.omg.org

[8] *Lightweight Directory Access Protocol*
http://directory.google.com/Top/Computers/Software/Internet/Servers/Directory/LDAP

[9] *Understanding and Deploying LDAP Directory services*
Timothy Howes et. al. ISBN 1-57870-070-1, MacMillan Technical publishing.

4 Metatype Specification

Version 1.0

4.1 Introduction

The Metatype specification defines interfaces that allow bundle developers to describe attribute types in a computer readable form using so-called *metadata*.

The purpose of this specification is to allow services to specify the type information of data that they can use as arguments. The data is based on *attributes*, which are key/value pairs like properties.

A designer in a type-safe language like Java is often confronted with the choice of using the language constructs to exchange data or using a technique based on attributes/properties that are based on key/value pairs. Attributes provide an escape from the rigid type-safety requirements of modern programming languages.

Type-safety works very well for software development environments in which multiple programmers work together on large applications or systems, but often lacks the flexibility needed to receive structured data from the outside world.

The attribute paradigm has several characteristics that make this approach suitable when data needs to be communicated between different entities which "speak" different languages. Attributes are uncomplicated, resilient to change, and allow the receiver to dynamically adapt to different types of data.

As an example, the OSGi Service Platform Specifications define several attribute types which are used in a Framework implementation, but which are also used and referenced by other OSGi specifications such as the *Configuration Admin Service Specification* on page 23. A Configuration Admin service implementation deploys attributes (key/value pairs) as configuration properties.

During the development of the Configuration Admin service, it became clear that the Framework attribute types needed to be described in a computer readable form. This information (the metadata) could then be used to automatically create user interfaces for management systems or could be translated into management information specifications such as CIM, SNMP, and the like.

4.1.1 Essentials

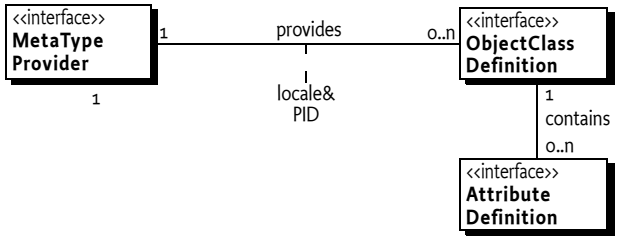
- *Conceptual model* – The specification must have a conceptual model for how classes and attributes are organized.

- *Standards* – The specification should be aligned with appropriate standards, and explained in situations where the specification is not aligned with, or cannot be mapped to, standards.
- *Remote Management* – Remote management should be taken into account.
- *Size* – Minimal overhead in size for a bundle using this specification is required.
- *Localization* – It must be possible to use this specification with different languages at the same time. This ability allows servlets to serve information in the language selected in the browser.
- *Type information* – The definition of an attribution should contain the name (if it is required), the cardinality, a label, a description, labels for enumerated values, and the Java class that should be used for the values.
- *Validation* – It should be possible to validate the values of the attributes.

4.1.2 Entities

- *Attribute* – A key/value pair.
- *AttributeDefinition* – Defines a description, name, help text, and type information of an attribute.
- *ObjectClassDefinition* – Defines the type of a datum. It contains a description and name of the type plus a set of AttributeDefinition objects.
- *MetaTypeProvider* – Provides access to the object classes that are available for this object. Access uses the PID and a locale to find the best ObjectClassDefinition object.

Figure 16 *Class Diagram Meta Typing, org.osgi.service.metatyping*



4.1.3 Operation

This specification starts with an object that implements the MetaTypeProvider interface. It is not specified how this object is obtained, and there are several possibilities. Often, however, this object is a service registered with the Framework.

A MetaTypeProvider object provides access to ObjectClassDefinition objects. These objects define all the information for a specific *object class*. An object class is a some descriptive information and a set of named attributes (which are key/value pairs).

Access to object classes is qualified by a locale and a Persistent IDentity (PID). The locale is a String object that defines for which language the ObjectClassDefinition is intended, allowing for localized user interfaces. The PID is used when a single MetaTypeProvider object can provide ObjectClassDefinition objects for multiple purposes. The context in which the MetaTypeProvider object is used should make this clear.

Attributes have global scope. Two object classes can consist of the same attributes, and attributes with the same name should have the same definition. This global scope is unlike languages like Java that scope instance variables within a class, but it is similar to the Lightweight Directory Access Protocol (LDAP) (SNMP also uses a global attribute name-space).

Attribute Definition objects provide sufficient localized information to generate user interfaces.

4.2 Attributes Model

The Framework uses the LDAP filter syntax for searching the Framework registry. The usage of the attributes in this specification and the Framework specification closely resemble the LDAP attribute model. Therefore, the names used in this specification have been aligned with LDAP. Consequently, the interfaces which are defined by this Specification are:

- AttributeDefinition
- ObjectClassDefinition
- MetaTypeProvider

These names correspond to the LDAP attribute model. For further information on ASN.1-defined attributes and X.500 object classes and attributes, see [11] *Understanding and Deploying LDAP Directory services.*

The LDAP attribute model assumes a global name-space for attributes, and object classes consist of a number of attributes. So, if an object class inherits the same attribute from different parents, only one copy of the attribute must become part of the object class definition. This name-space implies that a given attribute, for example cn, should *always* be the common name and the type must always be a String. An attribute cn cannot be an Integer in another object class definition. In this respect, the OSGi approach towards attribute definitions is comparable with the LDAP attribute model.

4.3 Object Class Definition

The ObjectClassDefinition interface is used to group the attributes which are defined in AttributeDefinition objects.

An ObjectClassDefinition object contains the information about the overall set of attributes and has the following elements:

- A name which can be returned in different locales.
- A global name-space in the registry, which is the same condition as LDAP/X.500 object classes. In these standards the OSI Object Identifier (OID) is used to uniquely identify object classes. If such an OID exists, (which can be requested at several standard organizations, and many

companies already have a node in the tree) it can be returned here. Otherwise, a unique id should be returned. This id can be a Java class name (reverse domain name) or can be generated with a GUID algorithm. All LDAP-defined object classes already have an associated OID. It is strongly advised to define the object classes from existing LDAP schemes which provide many preexisting OIDs. Many such schemes exist ranging from postal addresses to DHCP parameters.
- A human-readable description of the class.
- A list of attribute definitions which can be filtered as required, or optional. Note that in X.500 the mandatory or required status of an attribute is part of the object class definition and not of the attribute definition.
- An icon, in different sizes.

4.4 Attribute Definition

The AttributeDefinition interface provides the means to describe the data type of attributes.

The AttributeDefinition interface defines the following elements:

- Defined names (final ints) for the data types as restricted in the Framework for the attributes, called the syntax in OSI terms, which can be obtained with the getType() method.
- AttributeDefinition objects should use and ID that is similar to the OID as described in the ID field for ObjectClassDefinition.
- A localized name intended to be used in user interfaces.
- A localized description that defines the semantics of the attribute and possible constraints, which should be usable for tooltips.
- An indication if this attribute should be stored as a unique value, a Vector, or an array of values, as well as the maximum cardinality of the type.
- The data type, as limited by the Framework service registry attribute types.
- A validation function to verify if a possible value is correct.
- A list of values and a list of localized labels. Intended for popup menus in GUIs, allowing the user to choose from a set.
- A default value. The return type of this is a String[ ]. For cardinality = zero, this return type must be an array of one String object. For other cardinalities, the array must not contain more than the absolute value of *cardinality* String objects. In that case, it may contain 0 objects.

4.5 Meta Type Provider

The MetaTypeProvider interface is used to access metatype information. It is used in management systems and run-time management. It supports locales so that the text used in AttributeDefinition and ObjectClassDefinition objects can be adapted to different locales.

The PID is given as an argument with the getObjectClassDefinition method so that a single MetaTypeProvider object can be used for different object classes with their own PIDs.

Locale objects are represented in String objects because not all profiles support Locale. The String holds the standard Locale presentation of:

<language> ["_" <country> ["_" <variation>]]

For example, "en", "nl_BE", "en_CA_posix".

4.6 Metatype Example

AttributeDefinition and ObjectClassDefinition classes are intended to be easy to use for bundles. This example shows a naive implementation for these classes (note that the get methods usages are not shown). Commercial implementations can use XML, Java serialization, or Java Properties for implementations. This example uses plain code to store the definitions.

The example first shows that the ObjectClassDefinition interface is implemented in the OCD class. The name is made very short because the class is used to instantiate the static structures. Normally many of these objects are instantiated very close to each other, and long names would make these lists of instantiations very long.

```
class OCD implements ObjectClassDefinition {
   String                    name;
   String                    id;
   String                    description;
   AttributeDefinition       required[];
   AttributeDefinition       optional[];

   public OCD(
      String name, String id, String description,
      AttributeDefinition required[],
      AttributeDefinition optional[]) {

      this.name = name;
      this.id = id;
      this.description = description;
      this.required = required;
      this.optional = optional;
   }
   .... All the get methods
}
```

The second class is the AD class that implements the AttributeDefinition interface. The name is short for the same reason as in OCD. Note the two different constructors to simplify the common case.

```
class AD implements AttributeDefinition {
   String                    name;
   String                    id;
   String                    description;
   int                       cardinality;
   int                       syntax;
```

```
    String[]                values;
    String[]                labels;
    String[]                deflt;

    public AD( String name, String id, String description,
      int syntax, int cardinality, String values[],
      String labels[], String deflt[]) {
        this.name           = name;
        this.id             = id;
        this.description    = description;
        this.cardinality    = cardinality;
        this.syntax         = syntax;
        this.values         = values;
        this.labels         = labels;
    }

    public AD( String name, String id, String description,
      int syntax)
    {
      this(name,id,description,syntax,0,null,null, null);
    }
    ... All the get methods and validate method
}
```

The last part is the example that implements a MetaTypeProvider class. Only one locale is supported, the US locale. The OIDs used in this example are the actual OIDs as defined in X.500.

```
public class Example implements MetaTypeProvider {
    final static AD cn = new AD(
      "cn",         "2.5.4.3", "Common name", AD.STRING);
    final static AD sn = new AD(
      "sn",         "2.5.4.4", "Sur name", AD.STRING);
    final static AD description = new AD(
      "description", "2.5.4.13","Description", AD.STRING);
    final static AD seeAlso = new AD(
      "seeAlso",    "2.5.4.34", "See Also", AD.STRING);
    final static AD telephoneNumber = new AD(
      "telephoneNumber", "2.5.4.20", "Tel nr", AD.STRING);
    final static AD userPassword = new AD(
      "userPassword", "2.5.4.3", "Password", AD.STRING);

    final static ObjectClassDefinition person = new OCD(
      "person", "2.5.6.6", "Defines a person",
        new AD[] { cn, sn },
        new AD[] { description, seeAlso,
          telephoneNumber, userPassword}
    );

    public ObjectClassDefinition getObjectClassDefinition(
      String pid, String locale) {
      return person;
    }
```

```
    public String[] getLocales() {
       return new String[] { "en_US" };
    }
 }
```

This code shows that the attributes are defined in AD objects as final static. The example groups a number of attributes together in an OCD object.

As can be seen from this example, the resource issues for using AttributeDefinition, ObjectClassDefinition and MetaTypeProvider classes are minimized.

4.7 Limitations

The OSGi MetaType specification is intended to be used for simple applications. It does not, therefore, support recursive data types, mixed types in arrays/vectors, or nested arrays/vectors.

4.8 Related Standards

One of the primary goals of this specification is to make metatype information available at run-time with minimal overhead. Many related standards are applicable to metatypes; except for Java beans, however, all other metatype standards are based on document formats (e.g. XML). In the OSGi Service Platform, document format standards are deemed unsuitable due to the overhead required in the execution environment (they require a parser during run-time).

Another consideration is the applicability of these standards. Most of these standards were developed for management systems on platforms where resources are not necessarily a concern. In this case, a metatype standard is normally used to describe the data structures needed to control some other computer via a network. This other computer, however, does not require the metatype information as it is *implementing* this information.

In some traditional cases, a management system uses the metatype information to control objects in an OSGi Service Platform. Therefore, the concepts and the syntax of the metatype information must be mappable to these popular standards. Clearly, then, these standards must be able to describe objects in an OSGi Service Platform. This ability is usually not a problem, because the metatype languages used by current management systems are very powerful.

4.8.1 Beans

The intention of the Beans packages in Java comes very close to the metatype information needed in the OSGi Service Platform. The java.beans.- packages cannot be used, however, for the following reasons:

- Beans packages require a large number of classes that are likely to be optional for an OSGi Service Platform.

- Beans have been closely coupled to the graphic subsystem (AWT) and applets. Neither of these packages is available on an OSGi Service Platform.
- Beans are closely coupled with the type-safe Java classes. The advantage of attributes is that no type-safety is used, allowing two parties to have an independent versioning model (no shared classes).
- Beans packages allow all possible Java objects, not the OSGi subset as required by this specification.
- Beans have no explicit localization.
- Beans have no support for optional attributes.

4.9 Security Considerations

Special security issues are not applicable for this specification.

4.10 Changes

This specification has not been changed since the previous release.

4.11 org.osgi.service.metatype

The OSGi Metatype Package. Specification Version 1.1.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.metatype; specification-version=1.1
```

4.11.1 Summary

- AttributeDefinition - An interface to describe an attribute. [p.72]
- MetaTypeInformation - A MetaType Information object is created by the MetaTypeService to return meta type information for a specific bundle. [p.75]
- MetaTypeProvider - Provides access to metatypes. [p.76]
- MetaTypeService - The MetaType Service can be used to obtain meta type information for a bundle. [p.76]
- ObjectClassDefinition - Description for the data type information of an objectclass. [p.67]

4.11.2 public interface AttributeDefinition

An interface to describe an attribute.

An `AttributeDefinition` object defines a description of the data type of a property/attribute.

4.11.2.1 public static final int BIGDECIMAL = 10

The `BIGDECIMAL` (10) type. Attributes of this type should be stored as `BigDecimal`, `Vector` with `BigDecimal` or `BigDecimal[]` objects depending on `getCardinality()`.

Deprecated Since 1.1

4.11.2.2 public static final int BIGINTEGER = 9

The BIGINTEGER (9) type. Attributes of this type should be stored as BigInteger, Vector with BigInteger or BigInteger [] objects, depending on the getCardinality() value.

Deprecated Since 1.1

4.11.2.3 public static final int BOOLEAN = 11

The BOOLEAN (11) type. Attributes of this type should be stored as Boolean, Vector with Boolean or boolean[] objects depending on getCardinality().

4.11.2.4 public static final int BYTE = 6

The BYTE (6) type. Attributes of this type should be stored as Byte, Vector with Byte or byte[] objects, depending on the getCardinality() value.

4.11.2.5 public static final int CHARACTER = 5

The CHARACTER (5) type. Attributes of this type should be stored as Character, Vector with Character or char[] objects, depending on the getCardinality() value.

4.11.2.6 public static final int DOUBLE = 7

The DOUBLE (7) type. Attributes of this type should be stored as Double, Vector with Double or double[] objects, depending on the getCardinality() value.

4.11.2.7 public static final int FLOAT = 8

The FLOAT (8) type. Attributes of this type should be stored as Float, Vector with Float or float[] objects, depending on the getCardinality() value.

4.11.2.8 public static final int INTEGER = 3

The INTEGER (3) type. Attributes of this type should be stored as Integer, Vector with Integer or int[] objects, depending on the getCardinality() value.

4.11.2.9 public static final int LONG = 2

The LONG (2) type. Attributes of this type should be stored as Long, Vector with Long or long[] objects, depending on the getCardinality() value.

4.11.2.10 public static final int SHORT = 4

The SHORT (4) type. Attributes of this type should be stored as Short, Vector with Short or short[] objects, depending on the getCardinality() value.

4.11.2.11 public static final int STRING = 1

The STRING (1) type.

Attributes of this type should be stored as String, Vector with String or String[] objects, depending on the getCardinality() value.

4.11.2.12 public int getCardinality( )

- Return the cardinality of this attribute. The OSGi environment handles multi valued attributes in arrays ([]) or in `Vector` objects. The return value is defined as follows:

```
    x = Integer.MIN_VALUE    no limit, but use Vector
    x < 0                    -x = max occurrences, store in
Vector
    x > 0                     x = max occurrences, store in
array []
    x = Integer.MAX_VALUE    no limit, but use array []
    x = 0                     1 occurrence required
```

4.11.2.13 public String[] getDefaultValue( )

- Return a default for this attribute. The object must be of the appropriate type as defined by the cardinality and `getType()`. The return type is a list of `String` objects that can be converted to the appropriate type. The cardinality of the return array must follow the absolute cardinality of this type. E.g. if the cardinality = 0, the array must contain 1 element. If the cardinality is 1, it must contain 0 or 1 elements. If it is -5, it must contain from 0 to max 5 elements. Note that the special case of a 0 cardinality, meaning a single value, does not allow arrays or vectors of 0 elements.

Returns Return a default value or `null` if no default exists.

4.11.2.14 public String getDescription( )

- Return a description of this attribute. The description may be localized and must describe the semantics of this type and any constraints.

Returns The localized description of the definition.

4.11.2.15 public String getID( )

- Unique identity for this attribute. Attributes share a global namespace in the registry. E.g. an attribute `cn` or `commonName` must always be a `String` and the semantics are always a name of some object. They share this aspect with LDAP/X.500 attributes. In these standards the OSI Object Identifier (OID) is used to uniquely identify an attribute. If such an OID exists, (which can be requested at several standard organisations and many companies already have a node in the tree) it can be returned here. Otherwise, a unique id should be returned which can be a Java class name (reverse domain name) or generated with a GUID algorithm. Note that all LDAP defined attributes already have an OID. It is strongly advised to define the attributes from existing LDAP schemes which will give the OID. Many such schemes exist ranging from postal addresses to DHCP parameters.

Returns The id or oid

4.11.2.16 public String getName( )

- Get the name of the attribute. This name may be localized.

Returns The localized name of the definition.

4.11.2.17 public String[] getOptionLabels( )

- Return a list of labels of option values.

The purpose of this method is to allow menus with localized labels. It is associated with `getOptionValues`. The labels returned here are ordered in the same way as the values in that method.

If the function returns `null`, there are no option labels available.

This list must be in the same sequence as the `getOptionValues()` method. I.e. for each index i in `getOptionLabels`, i in `getOptionValues()` should be the associated value.

For example, if an attribute can have the value male, female, unknown, this list can return (for dutch) `new String[] { "Man", "Vrouw", "Onbekend" }`.

Returns A list values

4.11.2.18 public String[] getOptionValues( )

- Return a list of option values that this attribute can take.

  If the function returns `null`, there are no option values available.

  Each value must be acceptable to validate() (return "") and must be a `String` object that can be converted to the data type defined by getType() for this attribute.

  This list must be in the same sequence as `getOptionLabels()`. I.e. for each index i in `getOptionValues`, i in `getOptionLabels()` should be the label.

  For example, if an attribute can have the value male, female, unknown, this list can return `new String[] { "male", "female", "unknown" }`.

Returns A list values

4.11.2.19 public int getType( )

- Return the type for this attribute.

  Defined in the following constants which map to the appropriate Java type. `STRING`,`LONG`,`INTEGER`, `CHAR`,`BYTE`,`DOUBLE`,`FLOAT`, `BOOLEAN`.

4.11.2.20 public String validate(String value)

value The value before turning it into the basic data type

- Validate an attribute in `String` form. An attribute might be further constrained in value. This method will attempt to validate the attribute according to these constraints. It can return three different values:

```
null                    no validation present
""                          no problems detected
"..."                       A localized description of why the
value is wrong
```

Returns `null`, "", or another string

4.11.3 public interface MetaTypeInformation extends MetaTypeProvider

A MetaType Information object is created by the MetaTypeService to return meta type information for a specific bundle.

4.11.3.1 public Bundle getBundle( )

- Return the bundle for which this object provides metatype information.

Returns Bundle for which this object provides metatype information.

4.11.3.2 public String[] getFactoryPids( )

- Return the Factory PIDs (for ManagedServices) for which ObjectClassDefinition information is available.

Returns Array of Factory PIDs.

4.11.3.3 public String[] getPids( )

- Return the PIDs (for ManagedServices) for which ObjectClassDefinition information is available.

Returns Array of PIDs.

4.11.4 public interface MetaTypeProvider

Provides access to metatypes.

4.11.4.1 public String[] getLocales( )

- Return a list of available locales. The results must be names that consists of language [_ country [_ variation]] as is customary in the `Locale` class.

Returns An array of locale strings or `null` if there is no locale specific localization can be found.

4.11.4.2 public ObjectClassDefinition getObjectClassDefinition(String id, String locale)

id The ID of the requested object class. This can be a pid or factory pid returned by getPids or getFactoryPids.

locale The locale of the definition or `null` for default locale.

- Returns an object class definition for the specified id localized to the specified locale.

  The locale parameter must be a name that consists of `language`["_" `country`["_" `variation`]] as is customary in the `Locale` class. This `Locale` class is not used because certain profiles do not contain it.

Returns A `ObjectClassDefinition` object.

Throws `IllegalArgumentException` – If the id or locale arguments are not valid

4.11.5 public interface MetaTypeService

The MetaType Service can be used to obtain meta type information for a bundle. The MetaType Service will examine the specified bundle for meta type documents and to create the returned MetaTypeInformation object.

4.11.5.1 public MetaTypeInformation getMetaTypeInformation(Bundle bundle)

bundle The bundle for which meta type information is requested.

- Return the MetaType information for the specified bundle.

Returns MetaTypeInformation object for the specified bundle.

4.11.6 public interface ObjectClassDefinition

Description for the data type information of an objectclass.

4.11.6.1 public static final int ALL = -1

Argument for `getAttributeDefinitions(int)`.

`ALL` indicates that all the definitions are returned. The value is -1.

4.11.6.2 public static final int OPTIONAL = 2

Argument for `getAttributeDefinitions(int)`.

`OPTIONAL` indicates that only the optional definitions are returned. The value is 2.

4.11.6.3 public static final int REQUIRED = 1

Argument for `getAttributeDefinitions(int)`.

`REQUIRED` indicates that only the required definitions are returned. The value is 1.

4.11.6.4 public AttributeDefinition[] getAttributeDefinitions(int filter)

filter `ALL,REQUIRED,OPTIONAL`

- Return the attribute definitions for this object class.

  Return a set of attributes. The filter parameter can distinguish between `ALL`, `REQUIRED` or the `OPTIONAL` attributes.

Returns An array of attribute definitions or `null` if no attributes are selected

4.11.6.5 public String getDescription( )

- Return a description of this object class. The description may be localized.

Returns The description of this object class.

4.11.6.6 public InputStream getIcon(int size) throws IOException

size Requested size of an icon, e.g. a 16x16 pixels icon then size = 16

- Return an `InputStream` object that can be used to create an icon from.

  Indicate the size and return an `InputStream` object containing an icon. The returned icon maybe larger or smaller than the indicated size.

  The icon may depend on the localization.

Returns An InputStream representing an icon or `null`

4.11.6.7 public String getID( )

- Return the id of this object class.

  `ObjectDefintion` objects share a global namespace in the registry. They share this aspect with LDAP/X.500 attributes. In these standards the OSI Object Identifier (OID) is used to uniquely identify object classes. If such an OID exists, (which can be requested at several standard organisations and many companies already have a node in the tree) it can be returned here. Otherwise, a unique id should be returned which can be a java class name

(reverse domain name) or generated with a GUID algorithm. Note that all LDAP defined object classes already have an OID associated. It is strongly advised to define the object classes from existing LDAP schemes which will give the OID for free. Many such schemes exist ranging from postal addresses to DHCP parameters.

Returns The id of this object class.

4.11.6.8 public String getName( )

□ Return the name of this object class. The name may be localized.

Returns The name of this object class.

4.12 References

[10] *LDAP.*
Available at http://directory.google.com/Top/Computers/Software/Internet/Servers/Directory/LDAP

[11] *Understanding and Deploying LDAP Directory services*
Timothy Howes et. al. ISBN 1-57870-070-1, MacMillan Technical publishing.

5 Service Component Runtime Specification

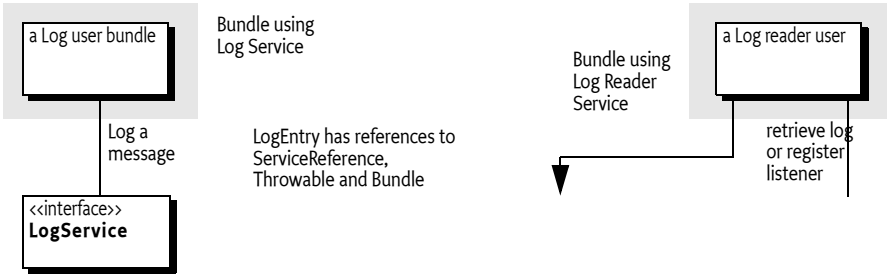
Version 1.0

5.1 Introduction

5.1.1 Entities

- *Application* – ....
- *Application Descrriptor* –

Figure 17 Log Service Class Diagram org.osgi.service.log package



5.2 The Service Component Runtime

5.3 Security

5.4 org.osgi.service.component

The OSGi Service Component Package. Specification Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.component; specification-version=1.0
```

5.4.1 Summary

- ComponentConstants - Defines standard names for Service Component constants. [p.80]

- ComponentContext - A ComponentContext interface is used by a Service Component to interact with it execution context including locating services by reference name. [p.80]
- ComponentException - Unchecked exception which may be thrown by the Service Component Runtime. [p.82]
- ComponentFactory - When a component is declared with the `factory` attribute on it's `component` element, the Service Component Runtime will register a ComponentFactory service to allow instances of the component to be created rather than automatically create component instances as necessary. [p.82]
- ComponentInstance - A ComponentInstance encapsulates an instance of a component. [p.83]

5.4.2 public interface ComponentConstants

Defines standard names for Service Component constants.

5.4.2.1 public static final String COMPONENT_FACTORY = "component.factory"

A service registration property for a Service Component Factory. It contains the value of the `factory` attribute. The type of this property must be `String`.

5.4.2.2 public static final String COMPONENT_NAME = "component.name"

A service registration property for a Service Component. It contains the name of the Service Component. The type of this property must be `String`.

5.4.2.3 public static final String REFERENCE_TARGET_SUFFIX = ".target"

A suffix for a service registration property for a reference target. It contains the filter to select the target services for a reference. The type of this property must be `String`.

5.4.2.4 public static final String SERVICE_COMPONENT = "Service-Component"

Manifest header (named "Service-Component") identifying the XML resources within the bundle containing the bundle's Service Component descriptions.

The attribute value may be retrieved from the `Dictionary` object returned by the `Bundle.getHeaders` method.

5.4.3 public interface ComponentContext

A ComponentContext interface is used by a Service Component to interact with it execution context including locating services by reference name. In order to be notified when a component is activated and to obtain a ComponentContext, the component's implementation class must implement a

```
protected void activate(ComponentContext context);
```

method. However, the component is not required to implement this method.

In order to be called when the component is deactivated, a component's implementation class must implement a

```
protected void deactivate(ComponentContext context);
```

method. However, the component is not required to implement this method.

These methods will be called by the Service Component Runtime using reflection and may be private methods to avoid being public methods on the component's provided service object.

5.4.3.1 public void disableComponent(String name)

name of a component.

- Disables the specified component name. The specified component name must be in the same bundle as this component.

5.4.3.2 public void enableComponent(String name)

name of a component or `null` to indicate all components in the bundle.

- Enables the specified component name. The specified component name must be in the same bundle as this component.

5.4.3.3 public BundleContext getBundleContext( )

- Returns the BundleContext of the bundle which contains this component.

Returns The BundleContext of the bundle containing this component.

5.4.3.4 public ComponentInstance getComponentInstance( )

- Returns the ComponentInstance object for this component.

Returns The ComponentInstance object for this component.

5.4.3.5 public Dictionary getProperties( )

- Returns the component properties for this ComponentContext.

Returns properties for this ComponentContext. The properties are read only and cannot be modified.

5.4.3.6 public Bundle getUsingBundle( )

- If the component is registered as a service using the `servicefactory="true"` attribute, then this method returns the bundle using the service provided by this component.

  This method will return `null` if the component is either:

  - Not a service, then no bundle can be using it as a service.
  - Is a service but did not specify the `servicefactory="true"` attribute, then all bundles will use this component.

Returns The bundle using this component as a service or `null`.

5.4.3.7 public Object locateService(String name)

name The name of a service reference as specified in a `reference` element in this component's description.

- Returns the service object for the specified service reference name.

Returns A service object for the referenced service or `null` if the reference cardinality is `0..1` or `0..n` and no matching service is available.

Throws `ComponentException` – If the Service Component Runtime catches an exception while activating the target service.

5.4.3.8 public Object[] locateServices(String name)

name The name of a service reference as specified in a `reference` element in this component's description.

- Returns the service objects for the specified service reference name.

Returns An array of service objects for the referenced service or `null` if the reference cardinality is `0..1` or `0..n` and no matching service is available.

Throws `ComponentException` – If the Service Component Runtime catches an exception while activating a target service.

5.4.4 public class ComponentException extends RuntimeException

Unchecked exception which may be thrown by the Service Component Runtime.

5.4.4.1 public ComponentException(String message, Throwable cause)

message The message for the exception.

cause The cause of the exception. May be `null`.

- Construct a new ComponentException with the specified message and cause.

5.4.4.2 public ComponentException(String message)

message The message for the exception.

- Construct a new ComponentException with the specified message.

5.4.4.3 public ComponentException(Throwable cause)

cause The cause of the exception. May be `null`.

- Construct a new ComponentException with the specified cause.

5.4.4.4 public Throwable getCause( )

- Returns the cause of this exception or `null` if no cause was specified when this exception was created.

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

5.4.4.5 public Throwable initCause(Throwable cause)

- The cause of this exception can only be set when constructed.

Throws `IllegalStateException` – This method will always throw an `IllegalStateException` since the cause of this exception can only be set when constructed.

5.4.5 public interface ComponentFactory

When a component is declared with the `factory` attribute on it's `component` element, the Service Component Runtime will register a ComponentFactory service to allow instances of the component to be created rather than automatically create component instances as necessary.

5.4.5.1 public ComponentInstance newInstance(Dictionary properties)

properties Additional properties for the component.

- Create a new instance of the component. Additional properties may be provided for the component instance.

Returns A ComponentInstance object encapsulating the component instance. The returned component instance has been activated.

5.4.6 public interface ComponentInstance

A ComponentInstance encapsulates an instance of a component. ComponentInstances are created whenever an instance of a component is created.

5.4.6.1 public void dispose( )

- Dispose of this component instance. The instance will be deactivated. If the instance has already been deactivated, this method does nothing.

5.4.6.2 public Object getInstance( )

- Returns the component instance. The instance has been activated.

Returns The component instance or `null` if the instance has been deactivated.

6 IO Connector Service Specification

Version 1.0

6.1 Introduction

Communication is at the heart of OSGi Service Platform functionality. Therefore, a flexible and extendable communication API is needed: one that can handle all the complications that arise out of the Reference Architecture. These obstacles could include different communication protocols based on different networks, firewalls, intermittent connectivity, and others.

Therefore, this IO Connector Service specification adopts the [12] *Java 2 Micro Edition* (J2ME) javax.microedition.io packages as a basic communications infrastructure. In J2ME, this API is also called the Connector framework. A key aspect of this framework is that the connection is configured by a single string, the URI.

In J2ME, the Connector framework can be extended by the vendor of the Virtual Machine, but cannot be extended at run-time by other code. Therefore, this specification defines a service that adopts the flexible model of the Connector framework, but allows bundles to extend the Connector Services into different communication domains.

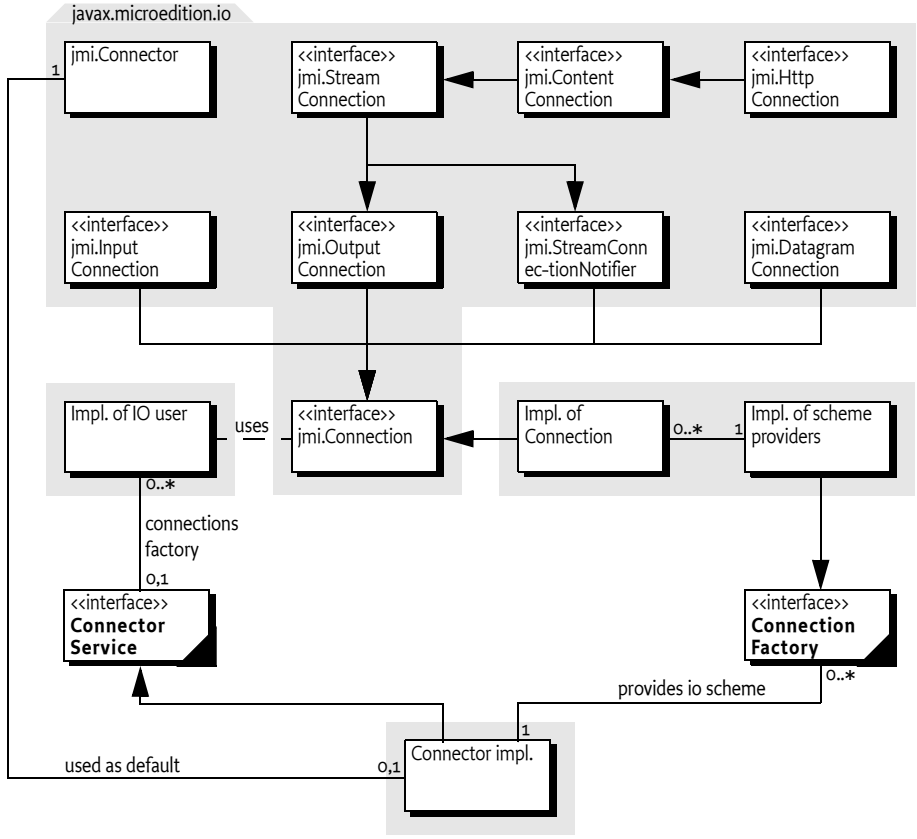
6.1.1 Essentials

- *Abstract* – Provide an intermediate layer that abstracts the actual protocol and devices from the bundle using it.
- *Extendable* – Allow third-party bundles to extend the system with new protocols and devices.
- *Layered* – Allow a protocol to be layered on top of lower layer protocols or devices.
- *Configurable* – Allow the selection of an actual protocol/device by means of configuration data.
- *Compatibility* – Be compatible with existing standards.

6.1.2 Entities

- *ConnectorService* – The service that performs the same function—creating connections from different providers—as the static methods in the Connector framework of javax.microediton.io.
- *ConnectionFactory* – A service that extends the Connector service with more schemes.
- *Scheme* – A protocol or device that is supported in the Connector framework.

Figure 18 Class Diagram, org.osgi.service.io (jmi is javax.microedition.io)



6.2 The Connector Framework

The [12] *Java 2 Micro Edition* specification introduces a package for communicating with back-end systems. The requirements for this package are very similar to the following OSGi requirements:

- Small footprint
- Allows many different implementations simultaneously
- Simple to use
- Simple configuration

The key design goal of the Connector framework is to allow an application to use a communication mechanism/protocol without understanding implementation details.

An application passes a Uniform Resource Identifier (URI) to the java.microedition.io.Connector class, and receives an object implementing one or more Connection interfaces. The java.microedition.io.Connector class uses the scheme in the URI to locate the appropriate Connection Factory service. The remainder of the URI may contain parameters that are used by the Connection Factory service to establish the connection; for example, they may contain the baud rate for a serial connection. Some examples:

- sms://+46705950899;expiry=24h;reply=yes;type=9
- datagram://:53
- socket://www.acme.com:5302
- comm://COM1;baudrate=9600;databits=9
- file:c:/autoexec.bat

The javax.microedition.io API itself does not prescribe any schemes. It is up to the implementor of this package to include a number of extensions that provide the schemes. The javax.microedition.io.Connector class dispatches a request to a class which provides an implementation of a Connection interface. J2ME does not specify how this dispatching takes place, but implementations usually offer a proprietary mechanism to connect user defined classes that can provide new schemes.

The Connector framework defines a taxonomy of communication mechanisms with a number of interfaces. For example, a javax.microedition.io.InputConnection interface indicates that the connection supports the input stream semantics, such as an I/O port. A javax.microedition.io.DatagramConnection interface indicates that communication should take place with messages.

When a javax.microedition.io.Connector.open method is called, it returns a javax.microedition.io.Connection object. The interfaces implemented by this object define the type of the communication session. The following interfaces may be implemented:

- *HttpConnection* – A javax.microedition.io.ContentConnection with specific HTTP support.
- *DatagramConnection* – A connection that can be used to send and receive datagrams.
- *OutputConnection* – A connection that can be used for streaming output.
- *InputConnection* – A connection that can be used for streaming input.
- *StreamConnection* – A connection that is both input and output.
- *StreamConnectionNotifier* – Can be used to wait for incoming stream connection requests.
- *ContentConnection* – A javax.microedition.io.StreamConnection that provides information about the type, encoding, and length of the information.

Bundles using this approach must indicate to the Operator what kind of interfaces they expect to receive. The operator must then configure the bundle with a URI that contains the scheme and appropriate options that match the bundle's expectations. Well-written bundles are flexible enough to communicate with any of the types of javax.microedition.io.Connection interfaces they have specified. For example, a bundle should support javax.microedition.io.StreamConnection as well as javax.microedition.io.DatagramConnection objects in the appropriate direction (input or output).

The following code example shows a bundle that sends an alarm message with the help of the javax.microedition.io.Connector framework:

```
public class Alarm {
   String     uri;
   public Alarm(String uri) { this.uri = uri; }
   private void send(byte[] msg) {
```

```
        while ( true ) try {
           Connection  connection = Connector.open( uri );
           DataOutputStream    dout = null;
           if ( connection instanceof OutputConnection ) {
              dout = ((OutputConnection)
                 connection).openDataOutputStream();
             dout.write( msg );
           }
           else if (connection instanceof DatagramConnection) {
             DatagramConnection dgc =
                 (DatagramConnection) connection;
             Datagram datagram = dgc.newDatagram(
                 msg, msg.length );
             dgc.send( datagram );
           } else {
               error( "No configuration for alarm" );
               return;
           }
           connection.close();
         } catch( Exception e ) { ... }
      }
   }
```

6.3 Connector Service

The javax.microedition.io.Connector framework matches the requirements for OSGi applications very well. The actual creation of connections, however, is handled through static methods in the javax.microedition.io.Connector class. This approach does not mesh well with the OSGi service registry and dynamic life-cycle management.

This specification therefore introduces the Connector Service. The methods of the ConnectorService interface have the same signatures as the static methods of the javax.microedition.io.Connector class.

Each javax.microedition.io.Connection object returned by a Connector Service must implement interfaces from the javax.microedition.io package. Implementations must strictly follow the semantics that are associated with these interfaces.

The Connector Service must provide all the schemes provided by the exporter of the javax.microedition.io package. The Connection Factory services must have priority over schemes implemented in the Java run-time environment. For example, if a Connection Factory provides the http scheme and a built-in implementation exists, then the Connector Service must use the Connection Factory service with the http scheme.

Bundles that want to use the Connector Service should first obtain a ConnectorService service object. This object contains open methods that should be called to get a new javax.microedition.io.Connection object.

6.4 Providing New Schemes

The Connector Service must be able to be extended with the Connection Factory service. Bundles that can provide new schemes must register a ConnectionFactory service object.

The Connector Service must listen for registrations of new ConnectionFactory service objects and make the supplied schemes available to bundles that create connections.

Implementing a Connection Factory service requires implementing the following method:

- createConnection(String,int,boolean) – Creates a new connection object from the given URI.

The Connection Factory service must be registered with the IO_SCHEME property to indicate the provided scheme to the Connector Service. The value of this property must be a String[] object.

If multiple Connection Factory services register with the same scheme, the Connector Service should select the Connection Factory service with the highest value for the service.ranking service registration property, or if more than one Connection Factory service has the highest value, the Connection Factory service with the lowest service.id is selected.

The following example shows how a Connection Factory service may be implemented. The example will return a javax.microedition.io.InputConnection object that returns the value of the URI after removing the scheme identifier.

```
public class ConnectionFactoryImpl
   implements BundleActivator, ConnectionFactory {
      public void start( BundleContext context ) {
         Hashtable  properties = new Hashtable();
         properties.put( IO_SCHEME,
            new String[] { "data" } );
         context.registerService(
            ConnectorService.class.getName(),
            this, properties );
      }
      public void stop( BundleContext context ) {}

      public Connection createConnection(
         String uri, int mode, boolean timeouts  ) {
         return new DataConnection(uri);
      }
}

class DataConnection
   implements javax.microedition.io.InputConnection {
   String     uri;
   DataConnection( String uri ) {this.uri = uri;}
   public DataInputStream openDataInputStream()
      throws IOException {
```

```
      return new DataInputStream( openInputStream() );
   }

   public InputStream openInputStream() throws IOException {
      byte [] buf = uri.getBytes();
      return new ByteArrayInputStream(buf,5,buf.length-5);
   }
   public void close() {}
}
```

6.4.1 Orphaned Connection Objects

When a Connection Factory service is unregistered, it must close all Connection objects that are still open. Closing these Connection objects should make these objects unusable, and they should subsequently throw an IOException when used.

Bundles should not unnecessarily hang onto objects they retrieved from services. Implementations of Connection Factory services should program defensively and ensure that resource allocation is minimized when a Connection object is closed.

6.5 Execution Environment

The javax.microedition.io package is available in J2ME configurations/profiles, but is not present in J2SE, J2EE, and the OSGi minimum execution requirements.

Implementations of the Connector Service that are targeted for all environments should carry their own implementation of the javax.microedition.io package and export it.

6.6 Security

The OSGi Connector Service is a key service available in the Service Platform. A malicious bundle which provides this service can spoof any communication. Therefore, it is paramount that the ServicePermission[REGISTER,ConnectorService] is given only to a trusted bundle. ServicePermission[GET,ConnectorService] may be handed to bundles that are allowed to communicate to the external world.

ServicePermission[REGISTER,ConnectionFactory] should also be restricted to trusted bundles because they can implement specific protocols or access devices. ServicePermission[GET,ConnectionFactory] should be limited to trusted bundles that implement the Connector Service.

Implementations of Connection Factory services must perform all I/O operations within a privileged region. For example, an implementation of the sms: scheme must have permission to access the mobile phone, and should not require the bundle that opened the connection to have this permission. Normally, the operations need to be implemented in a doPrivileged method or in a separate thread.

If a specific Connection Factory service needs more detailed permissions than provided by the OSGi or Java 2, it may create a new specific Permission sub-class for its purpose.

6.7 org.osgi.service.io

The OSGi IO Connector Specification Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.io; specification-version=1.0, javax.microedition.io
```

6.7.1 Summary

- ConnectionFactory - A Connection Factory service is called by the implementation of the Connector Service to create `javax.microedition.io.Connection` objects which implement the scheme named by `IO_SCHEME`. [p.89]
- ConnectorService - The Connector Service should be called to create and open `javax.microedition.io.Connection` objects. [p.91]

6.7.2 public interface ConnectionFactory

A Connection Factory service is called by the implementation of the Connector Service to create `javax.microedition.io.Connection` objects which implement the scheme named by `IO_SCHEME`. When a `ConnectorService.open` method is called, the implementation of the Connector Service will examine the specified name for a scheme. The Connector Service will then look for a Connection Factory service which is registered with the service property `IO_SCHEME` which matches the scheme. The `createConnection`[p.91] method of the selected Connection Factory will then be called to create the actual `Connection` object.

6.7.2.1 public static final String IO_SCHEME = "io.scheme"

Service property containing the scheme(s) for which this Connection Factory can create `Connection` objects. This property is of type `String[]`.

6.7.2.2 public Connection createConnection(String name, int mode, boolean timeouts) throws IOException

name The full URI passed to the `ConnectorService.open` method

mode The mode parameter passed to the `ConnectorService.open` method

timeouts The timeouts parameter passed to the `ConnectorService.open` method

□ Create a new `Connection` object for the specified URI.

Returns A new `javax.microedition.io.Connection` object.

Throws `IOException` – If a `javax.microedition.io.Connection` object can not not be created.

6.7.3 public interface ConnectorService

The Connector Service should be called to create and open `javax.microedition.io.Connection` objects. When an `open*` method is called, the implementation of the Connector Service will examine the specified name for a scheme. The Connector Service will then look for a Connection Factory service which is registered with the service property `IO_SCHEME` which matches the scheme. The `createConnection` method of the selected Connection Factory will then be called to create the actual `Connection` object.

If more than one Connection Factory service is registered for a particular scheme, the service with the highest ranking (as specified in its `service.ranking` property) is called. If there is a tie in ranking, the service with the lowest service ID (as specified in its `service.id` property), that is the service that was registered first, is called. This is the same algorithm used by `BundleContext.getServiceReference`.

6.7.3.1 public static final int READ = 1

Read access mode.

See Also `javax.microedition.io.Connector.READ`

6.7.3.2 public static final int READ_WRITE = 3

Read/Write access mode.

See Also `javax.microedition.io.Connector.READ_WRITE`

6.7.3.3 public static final int WRITE = 2

Write access mode.

See Also `javax.microedition.io.Connector.WRITE`

6.7.3.4 public Connection open(String name) throws IOException

name The URI for the connection.

- Create and open a `Connection` object for the specified name.

Returns A new `javax.microedition.io.Connection` object.

Throws `IllegalArgumentException` – If a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` – If the connection cannot be found.

`IOException` – If some other kind of I/O error occurs.

See Also `javax.microedition.io.Connector.open(String name)`

6.7.3.5 public Connection open(String name, int mode) throws IOException

name The URI for the connection.

mode The access mode.

- Create and open a `Connection` object for the specified name and access mode.

Returns A new `javax.microedition.io.Connection` object.

Throws `IllegalArgumentException` – If a parameter is invalid.

javax.microedition.io.ConnectionNotFoundException – If the connection cannot be found.

IOException – If some other kind of I/O error occurs.

See Also javax.microedition.io.Connector.open(String name, int mode)

6.7.3.6 public Connection open(String name, int mode, boolean timeouts) throws IOException

name The URI for the connection.

mode The access mode.

timeouts A flag to indicate that the caller wants timeout exceptions.

- Create and open a Connection object for the specified name, access mode and timeouts.

Returns A new javax.microedition.io.Connection object.

Throws IllegalArgumentException – If a parameter is invalid.

javax.microedition.io.ConnectionNotFoundException – If the connection cannot be found.

IOException – If some other kind of I/O error occurs.

See Also javax.microedition.io.Connector.open(String name, int mode, boolean timeouts)

6.7.3.7 public DataInputStream openDataInputStream(String name) throws IOException

name The URI for the connection.

- Create and open a DataInputStream object for the specified name.

Returns A DataInputStream object.

Throws IllegalArgumentException – If a parameter is invalid.

javax.microedition.io.ConnectionNotFoundException – If the connection cannot be found.

IOException – If some other kind of I/O error occurs.

See Also javax.microedition.io.Connector.openDataInputStream(String name)

6.7.3.8 public DataOutputStream openDataOutputStream(String name) throws IOException

name The URI for the connection.

- Create and open a DataOutputStream object for the specified name.

Returns A DataOutputStream object.

Throws IllegalArgumentException – If a parameter is invalid.

javax.microedition.io.ConnectionNotFoundException – If the connection cannot be found.

IOException – If some other kind of I/O error occurs.

See Also `javax.microedition.io.Connector.openDataOutputStream(String name)`

6.7.3.9 public InputStream openInputStream(String name) throws IOException

name The URI for the connection.

- Create and open an `InputStream` object for the specified name.

Returns An `InputStream` object.

Throws `IllegalArgumentException` – If a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` – If the connection cannot be found.

`IOException` – If some other kind of I/O error occurs.

See Also `javax.microedition.io.Connector.openInputStream(String name)`

6.7.3.10 public OutputStream openOutputStream(String name) throws IOException

name The URI for the connection.

- Create and open an `OutputStream` object for the specified name.

Returns An `OutputStream` object.

Throws `IllegalArgumentException` – If a parameter is invalid.

`javax.microedition.io.ConnectionNotFoundException` – If the connection cannot be found.

`IOException` – If some other kind of I/O error occurs.

See Also `javax.microedition.io.Connector.openOutputStream(String name)`

6.8 References

[12] *Java 2 Micro Edition*
http://java.sun.com/j2me/

[13] *javax.microedition.io whitepaper*
http://wireless.java.sun.com/midp/chapters/j2mewhite/chap13.pdf

[14] *J2ME Foundation Profile*
http://www.jcp.org/jsr/detail/46.jsp

7 Event Service Specification

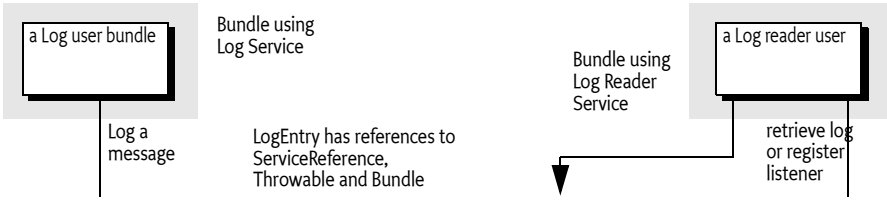
Version 1.0

7.1 Introduction

7.1.1 Entities

- *Application* – ....
- *Application Descrriptor* –

Figure 19 Log Service Class Diagram org.osgi.service.log package



7.2 The Event Service

7.3 Security

7.4 org.osgi.service.event

The OSGi Event Specification Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.event; specification-version=1.0
```

7.4.1 Summary

- ChannelEvent - [p.95]
- ChannelListener - [p.96]
- EventChannel - [p.96]

7.4.2 public class ChannelEvent

8 Deployment Admin Specification

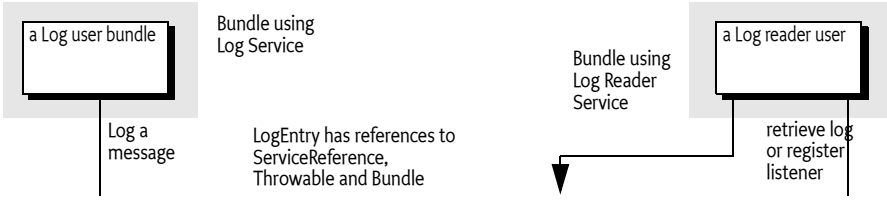
Version 1.0

8.1 Introduction

8.1.1 Entities

- *Application* – ....
- *Application Descrriptor* –

Figure 20 Log Service Class Diagram org.osgi.service.log package



8.2 The Deployment Admin Service

8.3 Security

.

8.4 org.osgi.service.deploymentadmin

The OSGi Deployment Admin Specification Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.deploymentadmin; specification-version=1.0
```

8.4.1 public interface DeploymentAdmin

TODO Add Javadoc comment for this type.

9 Application Model Service Specification

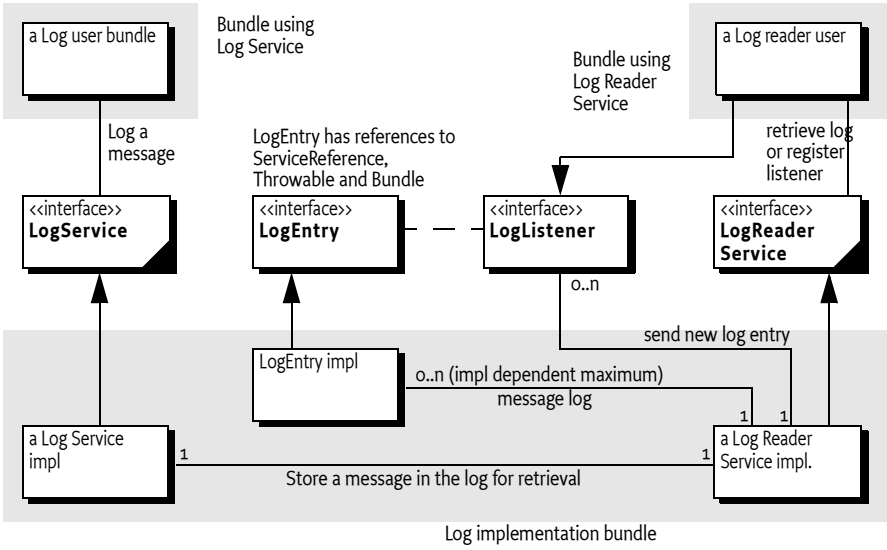
Version 1.0

9.1 Introduction

9.1.1 Entities

- *Application – ....*
- *Application Descrriptor –*

Figure 21 Log Service Class Diagram org.osgi.service.log package



9.2 The Application Admin Service Interface

9.3 Security

.

9.4 org.osgi.service.application

The OSGi Application Specification Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.application; specification-version=1.0
```

9.4.1 Summary

- Application - An application in the MEG environment. [p.100]
- ApplicationContainer - [p.100]
- ApplicationContext - [p.100]
- ApplicationDescriptor - Descriptor of an application. [p.100]
- ApplicationHandle - [p.101]
- ApplicationManager - [p.102]
- ApplicationManagerPermission - This class implements permissions for the Application Manager [p.104]
- MEGApplication - A MEG Application, a.k.a Meglet [p.104]
- MEGApplicationContext - [p.104]
- ScheduledApplication - A scheduled application contains the information of future registered applications. [p.105]
- SingletonException - [p.105]

9.4.2 public interface Application

An application in the MEG environment.

9.4.2.1 public void resumeApplication( ) throws Exception

9.4.2.2 public void startApplication( ) throws Exception

9.4.2.3 public void stopApplication( ) throws Exception

9.4.2.4 public void suspendApplication( ) throws Exception

9.4.3 public interface ApplicationContainer

9.4.3.1 public Application createApplication(ApplicationContext appContext, ApplicationHandle appHandle) throws Exception

9.4.3.2 public ApplicationDescriptor[] installApplication(InputStream inputStream) throws IOException, Exception

9.4.3.3 public ApplicationDescriptor[] uninstallApplication(ApplicationDescriptor appDescriptor, boolean force) throws IOException, Exception

9.4.3.4 public ApplicationDescriptor[] upgradeApplication(ApplicationDescriptor appDescriptor, InputStream inputStream, boolean force) throws IOException, Exception

9.4.4 public interface ApplicationContext

9.4.4.1 public Map getLaunchArgs( )

9.4.5 public interface ApplicationDescriptor

Descriptor of an application. This is the Application representer in the service registry. The application descriptor will be passed to an application container for instance creation.

9.4.5.1 public String getCategory( )

- Return the category of the application.

  The following list of application types are predefined:

  - APPTYPE_GAMES
  - APPTYPE_MESSAGING
  - APPTYPE_OFFICETOOLS

9.4.5.2 public String getContainerID( )

- Get the application container type. The following container types are supported by default

  - MEG (?)
  - Midlet
  - Doja

9.4.5.3 public Map getContainerProperties(String locale)

9.4.5.4 public String getName( )

- Returns the display name of application

9.4.5.5 public Map getProperties(String locale)

9.4.5.6 public String getUniqueID( )

- Get the unique identifier of the descriptor.

9.4.5.7 public String getVersion( )

- Returns the version descriptor of the service.

9.4.6 public interface ApplicationHandle

9.4.6.1 public static final int NONEXISTENT = 5

The application instance does not exist. Either an instance with the ID is never created .

9.4.6.2 public static final int RESUMING = 3

The application instance is being resumed. Status 'resumed' is equivalent to status 'running'

9.4.6.3 public static final int RUNNING = 0

The application instance is running

9.4.6.4 public static final int STOPPING = 4

The application instance is being stopped. Status 'stopped' is equivalent to status 'nonexistent'

9.4.6.5 **public static final int SUSPENDED = 2**

The application instance has been suspended

9.4.6.6 **public static final int SUSPENDING = 1**

The application instance is being suspended

9.4.6.7 **public void destroyApplication( ) throws Exception**

9.4.6.8 **public ApplicationDescriptor getAppDescriptor( )**

9.4.6.9 **public int getAppStatus( )**

- Get the status (constants defined in the Application class) of the application instance

9.4.6.10 **public void resumeApplication( ) throws Exception**

9.4.6.11 **public void suspendApplication( ) throws Exception**

9.4.7 public interface ApplicationManager

9.4.7.1 **public ScheduledApplication addScheduledApplication(ApplicationDescriptor appDescriptor, Map arguments, Date date)**

appUID the unique identifier of the application to be launched

arguments the arguments to launch the application

date the time to launch the application

- Add an application to be scheduled at a specified time.

Returns the id for this scheduled launch or -1 if the request is ignored

Throws `IOException` – if cannot access scheduled application list

`ApplicationNotFoundException` – if the appUID is not found

9.4.7.2 **public ApplicationDescriptor getAppDescriptor(String appUID) throws Exception**

appUID Application UID

- Get an application descriptor with the application UID

Throws `Exception` – Application with the specified ID cannot be found

9.4.7.3 **public ApplicationDescriptor[] getAppDescriptors( )**

- Get the descriptors of all the installed applications.

9.4.7.4 **public ScheduledApplication[] getScheduledApplications( ) throws IOException**

- Get a list of scheduled applications.

Returns an integer array containing the ids of the scheduled launches. Returns an array of size zero if no application is scheduled.

Throws `IOException` – if cannot access scheduled application list

9.4.7.5 **public String[] getSupportedMimeTypes( )**

- Get a list of MIME types supported in the system.

Returns A list of MIME types supported in the system

9.4.7.6 public boolean isLocked(ApplicationDescriptor appDescriptor) throws Exception

appUID Application UID of the application.

- Returns a boolean indicating whether this application is locked or not.

Returns A boolean value true/false indicating whether the application is locked or not.

Throws `ApplicationNotFoundException` – Application with the specified ID cannot be found

9.4.7.7 public ApplicationHandle launchApplication(ApplicationDescriptor appDescriptor, Map args) throws SingletonException, Exception

args Arguments for the newly launched application

- Launches a new instance of an application.

  The following steps should be performedif the represented application is a Meglet:

  1. If the application is a singleton and already has a running instance then throws SingletonException.

  2. If the bundle of the application is not ACTIVE then it starts that bundle. If the bundle has dependencies then the underlying bundles should also be started. This step may throw BundleException.

  3. Creates a MegletContext, but the MegletContext.getApplicationObject() will return null at this moment

  4. Then creates a new instance of the developer implemented application (Meglet) using its parameterless constructor.

  5. Calls Meglet.startApplication() and passes the MegletContext and the arguments to it

  6. If the startApplication() succeeds then it makes the ApplicationObject available on MegletContext.getApplicationObject() then registeres the ApplicationObject

  7. Returns the registered ApplicationObject

Returns The registered ApplicaitonObject which represents the newly launched applicaiton instance

Throws `SingletonException` – if the call attempts to launch a second instance of a singleton applicaiton

`BundleException` – if starting the bundle(s) failed

9.4.7.8 public void lock(ApplicationDescriptor appDescriptor) throws Exception

appUID UID of the application being locked - if UID is null, entire device can be locked

lockValue A boolean value true/false indicating whether the application should be locked or not.

- Sets the lock state of the application or the device.

Throws `ApplicationNotFoundException` – Application with the specified ID cannot be found

9.4.7.9 public void unLock(ApplicationDescriptor appDescriptor) throws Exception

appUID - UID or null if unlocking all applications on the device

passcode Passcode (PIN) value that will enable unlocking of the device/application

- Unset the lock of a specified application or all the applications

Throws `ApplicationNotFoundException` – Application with the specified ID cannot be found

9.4.8 public class ApplicationManagerPermission extends BasicPermission

This class implements permissions for the Application Manager

9.4.8.1 public static final String CONTENT = “content”

9.4.8.2 public static final String ENUMERATE = “enumerate”

9.4.8.3 public static final String LAUNCH = “launch”

9.4.8.4 public static final String PROVIDE = “provide”

9.4.8.5 public static final String SCHEDULE = “schedule”

9.4.8.6 public ApplicationManagerPermission(String actions)

actions - read and write

- Constructs a ApplicationManagerPermission.

9.4.8.7 public ApplicationManagerPermission(String name, String actions)

name - name of the permission

actions - read and write

- Constructs a ApplicationManagerPermission.

9.4.9 public abstract class MEGApplication implements Application , ChannelListener

A MEG Application, a.k.a Meglet

9.4.9.1 public MEGApplication(MEGApplicationContext context)

9.4.9.2 public abstract void channelEvent(ChannelEvent event)

9.4.9.3 public abstract void resumeApplication( ) throws Exception

9.4.9.4 public abstract void startApplication( ) throws Exception

9.4.9.5 public abstract void stopApplication( ) throws Exception

9.4.9.6 public abstract void suspendApplication( ) throws Exception

9.4.10 public interface MEGApplicationContext extends ApplicationContext

9.4.10.1 **public ApplicationManager getApplicationManager( )**

9.4.10.2 **public EventChannel getEventChannel( )**

9.4.10.3 **public Object getServiceObject(String className, String filter) throws Exception**

9.4.10.4 **public Object getServiceObject(String className, String filter, long millisecs) throws Exception**

9.4.10.5 **public boolean ungetServiceObject(Object serviceObject)**

9.4.11 public interface ScheduledApplication

A scheduled application contains the information of future registered applications.

9.4.11.1 **public ApplicationDescriptor getApplicationDescriptor( )**

9.4.11.2 **public Date getDate( )**

9.4.11.3 **public void remove( )**

9.4.12 public class SingletonException extends Exception

9.4.12.1 **public SingletonException( )**

10 Meglets Specification

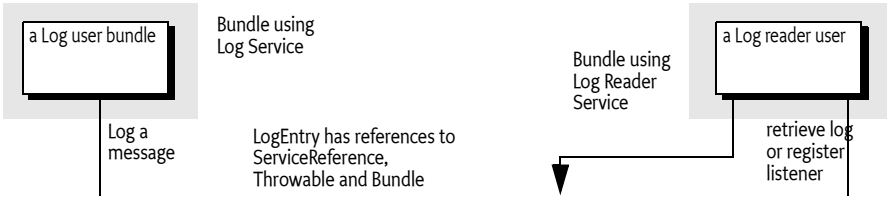
Version 1.0

10.1 Introduction

10.1.1 Entities

- *Application* – ....
- *Application Descrriptor* –

Figure 22 Log Service Class Diagram org.osgi.service.log package



10.2 The Meglet Base Class

10.3 Security

10.4 org.osgi.meglet

The OSGi Meglet Specification Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example:

```
Import-Package: org.osgi.meglet; specification-version=1.0
```

10.4.1 Summary

- Mobilet - A MEG Application, also called a Meglet. [p.107]
- MobiletFactory - [p.108]

10.4.2 public class Mobilet implements Application

A MEG Application, also called a Meglet.

10.4.2.1 public Mobilet( )

10.4.2.2 protected Dictionary getProperties( )

10.4.2.3 protected Object locateService(String name)

10.4.2.4 protected Object[] locateServices(String name)

10.4.2.5 protected void requestStop( )

10.4.2.6 protected void resume( ) throws Exception

10.4.2.7 public void resumeApplication( ) throws Exception

10.4.2.8 protected void start( ) throws Exception

10.4.2.9 public void startApplication(Map args) throws Exception

10.4.2.10 public void startApplication( ) throws Exception

Throws `Exception` –

See Also `org.osgi.service.application.Application.startApplication()`

10.4.2.11 protected void stop( ) throws Exception

10.4.2.12 public void stopApplication( ) throws Exception

10.4.2.13 protected void suspend( ) throws Exception

10.4.2.14 public void suspendApplication( ) throws Exception

10.4.3 public class MobiletFactory implements ComponentFactory

10.4.3.1 public MobiletFactory(ComponentFactory factory)

10.4.3.2 public ComponentInstance newInstance(Dictionary properties)

11 DMT Admin Service Specification

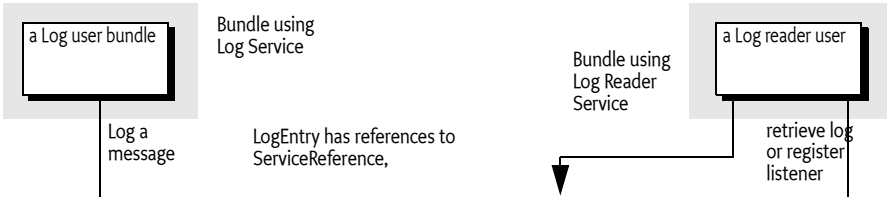
Version 1.0

11.1 Introduction

11.1.1 Entities

- *Application* – ....
- *Application Descrriptor* –

Figure 23 Log Service Class Diagram org.osgi.service.log package



11.2 The DMT Admin Service

11.3 Security

11.4 org.osgi.service.dmt

Unexpected tag, assume para (x=81,y=4) [p.123]

The DmtFactory interface is used to create DmtSession objects. The implementation of DmtFactory should register itself in the OSGi service registry as a service. DmtFactory is the entry point for applications to use the Dmt API. The <code>getTree</code> methods are used to open a session on a specified subtree of the DMT. A typical way of usage: <blockquote> <pre> serviceRef = context.getServiceReference(DmtFactory.class.getName()); DmtFactory factory = (DmtFactory) context.getService(serviceRef); DmtSession session = factory.getTree(null); session.createInteriorNode("./OSGi/cfg/mycfg"); </pre> </blockquote>

Invalid tag on top level blockquote (x=27,y=13) [p.123]

The DmtFactory interface is used to create DmtSession objects. The implementation of DmtFactory should register itself in the OSGi service registry as a service. DmtFactory is the entry point for applications to use the Dmt API. The ‹code›getTree‹/code› methods are used to open a session on a specified subtree of the DMT. A typical way of usage: ‹blockquote› ‹pre› serviceRef = context.getServiceReference(DmtFactory.class.getName()); DmtFactory factory = (DmtFactory) context.getService(serviceRef); DmtSession session = factory.getTree(null); session.createInteriorNode("./OSGi/cfg/mycfg"); ‹/pre› ‹/blockquote›

The OSGi DMT Specification Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.dmt; specification-version=1.0
```

11.4.1 Summary

- Dmt - A collection of DMT manipulation methods. [p.110]
- DmtAcl - The DmtAcl class represents structured access to DMT ACLs. [p.112]
- DmtAlertItem - Data structure carried in an alert (client initiated notification). [p.114]
- DmtAlertSender - The Dmt Admin provides the DmtAlertSender service which can be used by clients to send notifications to the server. [p.115]
- DmtData - A data structure representing a leaf node's value. [p.116]
- DmtDataPlugIn - An implementation of this interface takes the responsibility over a modifiable subtree of the DMT. [p.119]
- DmtDataType - A collection of constants describing the possible formats of a DMT node. [p.120]
- DmtEvent - [p.120]
- DmtException - Checked exception received when a DMT operation fails. [p.120]
- DmtExecPlugIn - An implementation of this interface takes the responsibility of handling EXEC requests in a subtree of the DMT. [p.122]
- DmtFactory - The DmtFactory interface is used to create DmtSession objects. [p.123]
- DmtMetaNode - The DmtMetaNode contains meta data both standard for SyncML DM and defined by OSGi MEG (without breaking the compatibility) to provide for better DMT data quality in an environment where many software components manipulate this data. [p.124]
- DmtPrincipal - The representation of an entity having the role of manipulating the Device Management Tree. [p.126]
- DmtReadOnly - This interface collects basic DMT operations. [p.127]
- DmtReadOnlyDataPlugIn - An implementation of this interface takes the responsibility over a non-modifiable subtree of the DMT. [p.128]
- DmtSession - DmtSession provides concurrent access to the DMT. [p.129]

11.4.2 public interface Dmt extends DmtReadOnly

A collection of DMT manipulation methods. The application programmers use these methods when they are interacting with a DmtSession which inherits from this interface. Data plugins also implement this interface.

11.4.2.1 public void clone(String nodeUri, String newNodeUri, boolean recursive) throws DmtException

nodeUri The node or root of a subtree to be copied

newNodeUri The URI of the new node or root of a subtree

recursive false if only a single node is copied, true if the whole subtree is copied.

- Create a deep copy of a node. All properties and values will be copied. The command works for single nodes and recursively for whole subtrees.

Throws `DmtException` –

11.4.2.2 public void createInteriorNode(String nodeUri) throws DmtException

nodeUri The URI of the node

- Create an interior node

Throws `DmtException` –

11.4.2.3 public void createInteriorNode(String nodeUri, String type) throws DmtException

nodeUri The URI of the node

type The type URL of the interior node

- Create an interior node with a given type. The type of interior node is an URL pointing to a DDF document.

Throws `DmtException` –

11.4.2.4 public void createLeafNode(String nodeUri, DmtData value) throws DmtException

nodeUri The URI of the node

value The value to be given to the new node or null.

- Create a leaf node with a given value. The default constructor for DmtData is used to create nodes with default values. If a node does not have a default value, this method using such a DmtData object will throw a DmtException with error code METADATA_MISMATCH. If null is passed as the second parameter of the method, a node with null format will be created.

Throws `DmtException` –

11.4.2.5 public void deleteNode(String nodeUri) throws DmtException

nodeUri The URI of the node

- Delete the given node. Deleting interior nodes is recursive, the whole subtree under the given node is deleted.

Throws `DmtException` –

11.4.2.6 public void renameNode(String nodeUri, String newName) throws DmtException

nodeUri The URI of the node to rename

newName The new name property of the node. This is not the new URI of the node, the new URI is constructed from the old URI and the new name.

- Rename a node. The value and the other properties of the node does not change.

Throws `DmtException` –

11.4.2.7 public void rollback( ) throws DmtException

- Rolls back a series of DMT operations issued in the current session since it was opened.

Throws `DmtException` – The error code is ROLLBACK_FAILED in case the rollback did not succedd. FEATURE_NOT_SUPPORTED in case the session was not created using the LOCK_TYPE_ATOMIC lock type.

11.4.2.8 public void setNodeTitle(String nodeUri, String title) throws DmtException

nodeUri The URI of the node

title The title text of the node

- Set the title property of a node.

Throws `DmtException` –

11.4.2.9 public void setNodeType(String nodeUri, String type) throws DmtException

nodeUri The URI of the node

type The type of the node

- Get the type of a node. The type of leaf node is the MIME type of the data it contains. The type of interior node is an URL pointing to a DDF document.

Throws `DmtException` –

11.4.2.10 public void setNodeValue(String nodeUri, DmtData data) throws DmtException

nodeUri The URI of the node

data The data to be set. The format of the node is contained in the DmtData. If null is given, the node will have the OMA null format.

- Set the value of a leaf node.

Throws `DmtException` –

11.4.3 public class DmtAcl

The DmtAcl class represents structured access to DMT ACLs. Under OMA DM the ACLs are defined as strings with an internal syntax.

The methods of this class taking a principal as parameter accept remote server IDs (as passed to DmtFactory.getTree), as well as "∗" indicating any principal.

11.4.3.1 public static final int ADD = 2

Principals holding this permission can issue ADD commands on the node having this ACL.

11.4.3.2 public static final int DELETE = 8

Principals holding this permission can issue DELETE commands on the node having this ACL.

11.4.3.3 public static final int EXEC = 16

Principals holding this permission can issue EXEC commands on the node having this ACL.

11.4.3.4 public static final int GET = 1

Principals holding this permission can issue GET command on the node having this ACL.

11.4.3.5 public static final int REPLACE = 4

Principals holding this permission can issue REPLACE commands on the node having this ACL.

11.4.3.6 public DmtAcl( )

- Create an instance of the ACL that represents an empty list of principals with no permissions.

11.4.3.7 public DmtAcl(String acl)

acl The string representation of the ACL as defined in OMA DM. If null then it represents an empty list of principals with no permissions.

- Create an instance of the ACL from its canonic string representation.

Throws `IllegalArgumentException` – if acl is not a valid OMA DM ACL string

11.4.3.8 public DmtAcl(DmtAcl acl)

- Creates an instance of the ACL that represents the same permissions as the parameter ACL object.

Throws `IllegalArgumentException` – if the given ACL object is not consistent (e.g. some principals do not have all the global permissions), or if the parameter changes during the call

11.4.3.9 public synchronized void addPermission(String principal, int permissions)

principal The entity to which permission should be granted.

permissions The permissions to be given. The parameter can be a logical or of more permission constants defined in this class.

- Add a specific permission to a given principal. The already existing permissions of the principal are not affected.

11.4.3.10 public synchronized void deletePermission(String principal, int

permissions)

principal The entity from which a permission should be revoked.

permissions The permissions to be revoked. The parameter can be a logical or of more permission constants defined in this class.

- Revoke a specific permission from a given principal. Other permissions of the principal are not affected.

11.4.3.11 public synchronized int getPermissions(String principal)

principal The entity whose permissions to query

- Get the permissions associated to a given principal.

Returns The permissions which the given principal has. The returned int is the logical or of the permission constants defined in this class.

11.4.3.12 public Vector getPrincipals( )

- Get the list of principals who have any kind of permissions on this node.

Returns The set of principals having permissions on this node.

11.4.3.13 public synchronized boolean isPermitted(String principal, int permissions)

principal The entity to check

permissions The permission to check

- Check whether a given permission is given to a certain principal.

Returns true if the principal holds the given permission

11.4.3.14 public synchronized void setPermission(String principal, int permissions)

principal The entity to which permission should be granted.

permissions The set of permissions to be given. The parameter can be a logical or of the permission constants defined in this class.

- Set the list of permissions a given principal has. All permissions the principal had will be overwritten.

11.4.3.15 public synchronized String toString( )

- Give the canonic string representation of this ACL.

Returns The string representation as defined in OMA DM.

11.4.4 public class DmtAlertItem

Data structure carried in an alert (client initiated notification). The DmtAlertItem describes details of various alerts that can be sent by the client of the OMA DM protocol. The use cases include the client sending a session request to the server (alert 1201), the client notifying the server of completion of a software update operation (alert 1226) or sending back results in response to an asynchronous EXEC command.

The data syntax and semantics varies widely between various alerts, so does the optionality of particular parameters of an alert item. If an item, such as source or type, is not defined, the corresponding getter method returns null. For example, for alert 1201 (client-initiated session) all elements will be null.

11.4.4.1 public DmtAlertItem(String source, String type, String format, String mark, String data)

source The URI of the node which is the source of the alert item

type The type of the alert item

format The format of the alert item

mark The markup of the alert item (as defined by OMA DM standards) If this is null, but at least one of the other parameters are not null, the markup string returned by the getMark() method is created using the other parameters. If this parameter is not null, the getMark() method returns this value.

data The data of the alert item

- Create an instance of the alert item. The constructor takes all possible data entries as parameters. Any of these parameters can be null

11.4.4.2 public String getData( )

- Get the data associated with the alert item. There might be no data associated with the alert item.

Returns The data associated with the alert item. Can be null.

11.4.4.3 public String getFormat( )

- Get the format associated with the alert item. There might be no format associated with the alert item.

Returns The format associated with the alert item. Can be null.

11.4.4.4 public String getMark( )

- Get the markup data associated with the alert item. There might be no markup associated with the alert item.

Returns The markup data associated with the alert item, in the format defined by OMA DM specifications. Can be null.

11.4.4.5 public String getSource( )

- Get the node which is the source of the alert. There might be no source associated with the alert item.

Returns The URI of the node which is the source of this alert. Can be null.

11.4.4.6 public String getType( )

- Get the type associated with the alert item. There might be no format associated with the alert item.

Returns The type type associated with the alert item. Can be null.

11.4.4.7 public String toString( )

11.4.5 public interface DmtAlertSender

The Dmt Admin provides the DmtAlertSender service which can be used by clients to send notifications to the server. The clients find the service through the OSGi service registry. The typical client of this service is a Dmt Plugin which send an alert asynchronously after an exec operation. It is the Dmt Admin's responsibility to route the alert to the appropriate protocol adaptor. Routing is possible based on the session or server information provided as a parameter in the sendAlert() methods.

11.4.5.1 public void sendAlert(DmtSession session, int code, DmtAlertItem[] items) throws DmtException

session The session what the plugin received in its open() method

code Alert code. Can be 0 if not needed.

items The data of the alert items carried in this alert

- Sends an alert, where routing is based on session information. The plugin receives a session reference in its open() method so it can provide this information to the AlertSender. The DmtAdmin uses the session information to find out the server ID where the notification must be sent. If the session was initiated locally the alert must not be sent. The session might be closed already at the time this method is called, however the DmtAdmin must still be able to deduce the server ID.

Throws `DmtException` – Thrown when the alert can not be routed to the server. The code of the error is DmtException.ALERT_NOT_ROUTED

11.4.5.2 public void sendAlert(String serverid, int code, DmtAlertItem[] items) throws DmtException

serverid The ID of the remote server

code Alert code. Can be 0 if not needed.

items The data of the alert items carried in this alert

- Sends an alert, where routing is based on a server ID. The client might know the ID of a server which it wants to notify, in this case this form of the method must be used. If OMA DM is used as a management protocol the server ID corresponds to a DMT node value in ./SyncML/DMAcc/x/ServerId.

Throws `DmtException` – Thrown when the alert can not be routed to the server. The code of the error is DmtException.ALERT_NOT_ROUTED

11.4.5.3 public void sendAlert(int code, DmtAlertItem[] items) throws DmtException

code Alert code. Can be 0 if not needed.

items The data of the alert items carried in this alert

- Sends an alert, when the client does not have any routing hints to provide. Even in this case the routing might be possible if the DmtAdmin is connected to only one protocol adapter which is connected to only one remote server.

Throws `DmtException` – Thrown when the alert can not be routed to the server. The code of the error is DmtException.ALERT_NOT_ROUTED

11.4.6 public class DmtData

A data structure representing a leaf node's value. Creating instances of DmtData can happen in two ways, either with specifying a MIME type or without.

When creating an instance without explicitly specifying a MIME type, MIME type information will be derived from either the old value of the node (if the DmtData is used to update an already existing node's value), or from the meta data associated with this node (in case the DmtData is used in a Dmt.createLeafNode() method). In the latter case the meta data should allow only one MIME type, otherwise Dmt.createLeafNode() will fail.

Similarly, if a constructor does not specify the value of the new DmtData instance, its setting is also deferred till the invocation of either Dmt.setNode() or Dmt.createLeafNode() method. If the corresponding meta node has a default value, it is set, otherwise a DmtException is thrown.

11.4.6.1 public static final String NULL_FORMAT = "_magic_null_value_"

11.4.6.2 public DmtData( )

- Create a DmtData instance of null format.

11.4.6.3 public DmtData(String str)

str The string value to set, or DmtData.NULL_FORMAT to create a node with the OMA DM null format

- Create a DmtData instance of chr or null format. If the parameter equals DmtData.NULL_FORMAT then a node with null format is created, otherwise a node with chr format is created and its value is set.

11.4.6.4 public DmtData(String str, boolean xml)

str The string or xml value to set

xml If true then a node of xml is created otherwise this constructor behaves the same as DmtData(String).

- Create a DmtData instance of xml format and set its value.

11.4.6.5 public DmtData(int integer)

integer The integer value to set

- Create a DmtData instance of int format and set its value.

11.4.6.6 public DmtData(boolean bool)

bool The boolean value to set

- Create a DmtData instance of bool format and set its value.

11.4.6.7 public DmtData(byte[] bytes)

bytes The byte array to set

- Create a DmtData instance of bin format and set its value.

11.4.6.8 public DmtData(String str, String mimeType)

str The string value to set

mimeType The MIME type to set

- Create a DmtData instance of chr format and set its value while also setting its MIME type.

11.4.6.9 public DmtData(String str, boolean xml, String mimeType)

str The string or xml value to set

xml If true then a node of xml is created otherwise this constructor behaves the same as DmtData(String,String).

mimeType The MIME type to set.

- Create a DmtData instance of xml format and set its value and also its MIME type.

11.4.6.10 public DmtData(byte[] bytes, String mimeType)

bytes The byte array to set

mimeType The MIME type to set

- Create a DmtData instance of bool format and set its value and also its MIME type.

11.4.6.11 public boolean equals(Object obj)

11.4.6.12 public byte[] getBinary( )

- Gets the value of a node with binary format

Returns The binary value

Throws `DmtException` – if the format of the node is not binary

11.4.6.13 public boolean getBoolean( ) throws DmtException

- Gets the value of a node with boolean format

Returns The boolean value

Throws `DmtException` – if the format of the node is not boolean

11.4.6.14 public int getFormat( )

- Get the node's format, expressed in terms of type constants defined in DmtDataType. Note that the 'format' term is a legacy from OMA DM, it is more customary to think of this as 'type'.

Returns The format of the node.

11.4.6.15 public int getInt( ) throws DmtException

- Gets the value of a node with integer format

Returns The integer value

Throws `DmtException` – if the format of the node is not boolean

11.4.6.16 public String getMimeType( )

- Get the current MIME type of the node.

Returns The MIME type string

11.4.6.17 public String getString( )

- Gets the string representation of the DmtNode. This method works for all formats. [TODO specify for all formats. what does it mean if binary]

Returns The string value of the DmtData

11.4.6.18 public int hashCode( )

11.4.6.19 public String toString( )

11.4.7 public interface DmtDataPlugIn extends Dmt

An implementation of this interface takes the responsibility over a modifiable subtree of the DMT. If the subtree is non modifiable then the DmtReadOnlyDataPlugin interface should be used.

The plugin might support transactionality, in this case it has to implemenent commit and rollback functionality.

In an OSGi environment such implementations should be registered at the OSGi service registry specifying the list of root node URIs in a String array in the dataRootURIs registration parameter.

11.4.7.1 public DmtMetaNode getMetaNode(String nodeUri, DmtMetaNode generic) throws DmtException

nodeUri The URI of the node of which metadata information is queried

generic Metadata information supplied by the Dmt Admin

- Get metadata information about a given node. The plugin implementation may complement the metadata supplied by the Dmt Admin by its own, plugin specific metadata. This mechanism can be used, for example, when a data plugin is implemented on top of a data store or another API that have their own metadata, such as a relational database, in order to avoid metadata duplication and inconsistency.

Returns The complete metadata information

Throws `DmtException` –

11.4.7.2 public void open(String subtreeUri, int lockMode, DmtSession session) throws DmtException

subtreeUri The subtree which is locked in the current session

lockMode One of the lock type constants specified in DmtSession

session The session from which this plugin instance is accessed

- This method is called to signal the start of a transaction when the first reference is made within a DmtSession to a node which is handled by this plugin.

Throws `DmtException` –

11.4.7.3 public boolean supportsAtomic( )

- Tells whether the plugin can handle atomic transactions. If a session is created using DmtSession.LOCK_TYPE_ATOMIC locking and the plugin supports it then it is possible to roll back operations in the session.

Returns true if the plugin can handle atomic transactions

11.4.8 public interface DmtDataType

A collection of constants describing the possible formats of a DMT node.

11.4.8.1 public static final int BINARY = 4

The node holds an OMA DM binary value. The value of the node corresponds to the Java byte[] type.

11.4.8.2 public static final int BOOLEAN = 3

The node holds an OMA DM bool value.

11.4.8.3 public static final int INTEGER = 1

The node holds an integer value. Note that this does not correspond to the Java int type, OMA DM integers are unsigned.

11.4.8.4 public static final int NODE = 7

The node is an internal node.

11.4.8.5 public static final int NULL = 6

The node holds an OMA DM null value. This corresponds to the Java null type.

11.4.8.6 public static final int STRING = 2

The node holds an OMA DM chr value.

11.4.8.7 public static final int XML = 5

The node holds an OMA DM xml value.

11.4.9 public interface DmtEvent

11.4.9.1 public static final int ADD = 1

11.4.9.2 public static final int DELETE = 3

11.4.9.3 public static final int REPLACE = 2

11.4.9.4 public Hashtable getNodes( )

11.4.9.5 public int getSessionId( )

11.4.9.6 public String getURI( )

11.4.10 public class DmtException extends Exception

Checked exception received when a DMT operation fails.

11.4.10.1 **public static final int ALERT_NOT_ROUTED = 5**

11.4.10.2 **public static final int COMMAND_FAILED = 500**

11.4.10.3 **public static final int COMMAND_NOT_ALLOWED = 405**

11.4.10.4 **public static final int CONCURRENT_ACCESS = 3**

11.4.10.5 **public static final int DATA_STORE_FAILURE = 510**

11.4.10.6 **public static final int DEVICE_FULL = 420**

11.4.10.7 **public static final int FEATURE_NOT_SUPPORTED = 406**

11.4.10.8 **public static final int FORMAT_NOT_SUPPORTED = 415**

11.4.10.9 **public static final int INVALID_DATA = 4**

11.4.10.10 **public static final int METADATA_MISMATCH = 2**

11.4.10.11 **public static final int NODE_ALREADY_EXISTS = 418**

11.4.10.12 **public static final int NODE_NOT_FOUND = 404**

11.4.10.13 **public static final int OTHER_ERROR = 0**

11.4.10.14 **public static final int PERMISSION_DENIED = 425**

11.4.10.15 **public static final int REMOTE_ERROR = 1**

11.4.10.16 **public static final int ROLLBACK_FAILED = 516**

11.4.10.17 **public static final int UNAUTHORIZED = 405**

11.4.10.18 **public static final int URI_TOO_LONG = 414**

11.4.10.19 **public DmtException(String uri, int code, String message)**

uri The node on which the failed DMT operation was issued

code The error code of the failure

message Message associated with the exception

- Create an instance of the exception. No originating exception is specified.

11.4.10.20 **public DmtException(String uri, int code, String message, Throwable cause)**

uri The node on which the failed DMT operation was issued

code The error code of the failure

message Message associated with the exception

cause The originating exception

- Create an instance of the exception, specifying the cause exception.

11.4.10.21 **public DmtException(String uri, int code, String message, Vector causes)**

uri The node on which the failed DMT operation was issued

code The error code of the failure

message Message associated with the exception

causes The list of originating exceptions

- Create an instance of the exception, specifying the list of cause exceptions.

11.4.10.22 public Throwable getCause( )

- Get the cause of this exception. Returns non- null, if this exception is caused by one or more other exceptions (like a NullPointerException in a Dmt Plugin).

11.4.10.23 public Vector getCauses( )

- Get all causes of this exception. Returns the causing exceptions in a vector. If no cause was specified, an empty vector is returned.

11.4.10.24 public int getCode( )

- Get the error code associated with this exception. Most of the error codes (returned by getCode()) within this exception correspond to OMA DM error codes.

Returns the error code

11.4.10.25 public String getMessage( )

- Get the message associated with this exception. The message also contains the associated URI and the exception code, if specified.

Returns the error message, or null if not specified

11.4.10.26 public String getURI( )

- Get the node on which the failed DMT operation was issued. Some operations like DmtSession.close() don't require an URI, in this case this method returns null.

Returns the URI of the node, or null

11.4.10.27 public void printStackTrace(PrintStream s)

11.4.10.28 public void printStackTrace(PrintWriter s)

11.4.11 public interface DmtExecPlugIn

An implementation of this interface takes the responsibility of handling EXEC requests in a subtree of the DMT.

In an OSGi environment such implementations should be registered at the OSGi service registry specifying the list of root node URIs in a String array in the execRootURIs registration parameter.

11.4.11.1 public void execute(DmtSession session, String nodeUri, String data) throws DmtException

session A reference to the session in which the operation was issued. Session information is needed in case an alert should be sent back from the plugin.

nodeUri The node to be executed.

data The data of the EXEC operation. The format of the data is not specified, it depends on the definition of the managed object (the node).

- Execute the given node with the given data. The execute() method of the DmtSession is forwarded to the appropriate DmtExecPlugin which handles the request. This operation corresponds to the EXEC command in OMA DM. The semantics of an EXEC operation and the data parameters it takes depend on the definition of the managed object on which the command is issued.

Throws DmtException –

11.4.12 public interface DmtFactory

The DmtFactory interface is used to create DmtSession objects. The implementation of DmtFactory should register itself in the OSGi service registry as a service. DmtFactory is the entry point for applications to use the Dmt API. The getTree methods are used to open a session on a specified subtree of the DMT. A typical way of usage:

```
 serviceRef = context.getServiceReference(DmtFacto-
ry.class.getName());
 DmtFactory factory = (DmtFactory) context.getService(service-
Ref);
 DmtSession session = factory.getTree(null);
 session.createInteriorNode(”./OSGi/cfg/mycfg”);
```

11.4.12.1 public DmtSession getTree(String principal) throws DmtException

principal the identifier of the remote server on whose behalf the data manipulation is performed, or null for local sessions

- Opens a DmtSession on the whole DMT as subtree. It is recommended to use other forms of this method where operations are issued only on a specific subtree. This call is equivalent to the following: getTree(principal, “.”, DmtSession.LOCK_TYPE_AUTOMATIC)

Returns a DmtSession object on which DMT manipulations can be performed

Throws DmtException –

11.4.12.2 public DmtSession getTree(String principal, String subtreeUri) throws DmtException

principal the identifier of the remote server on whose behalf the data manipulation is performed, or null for local sessions

subtreeUri The subtree on which DMT manipulations can be performed within the returned session

- Opens a DmtSession on a specific DMT subtree. This call is equivalent to the following: getTree(principal, subtreeUri, DmtSession.LOCK_TYPE_AUTOMATIC)

Returns a DmtSession object on which DMT manipulations can be performed

Throws DmtException –

11.4.12.3 public DmtSession getTree(String principal, String subtreeUri, int

lockMode) throws DmtException

principal the identifier of the remote server on whose behalf the data manipulation is performed, or null for local sessions

subtreeUri The subtree on which DMT manipulations can be performed within the returned session

lockMode One of the locking modes specified in DmtSession

- Opens a DmtSession on a specific DMT subtree using a specific locking mode.

Returns a DmtSession object on which DMT manipulations can be performed

Throws `DmtException` –

11.4.13 public interface DmtMetaNode

The DmtMetaNode contains meta data both standard for SyncML DM and defined by OSGi MEG (without breaking the compatibility) to provide for better DMT data quality in an environment where many software components manipulate this data.

The interface has two types of functions to describe type of nodes in the DMT. One is used to retrieve standard OMA DM metadata, such as access mode, cardinality, default etc. Another is used for meta data extensions defined by OSGi MEG, such as valid values, referential integrity (RI), regular expressions and such.

11.4.13.1 public boolean canAdd( )

- Check whether the ADD operation is valid for this node

Returns true if the operation is valid for this node

11.4.13.2 public boolean canDelete( )

- Check whether the DELETE operation is valid for this node

Returns true if the operation is valid for this node

11.4.13.3 public boolean canExecute( )

- Check whether the EXECUTE operation is valid for this node

Returns true if the operation is valid for this node

11.4.13.4 public boolean canGet( )

- Check whether the GET operation is valid for this node

Returns true if the operation is valid for this node

11.4.13.5 public boolean canReplace( )

- Check whether the REPLACE operation is valid for this node

Returns true if the operation is valid for this node

11.4.13.6 public String[] getChildURIs( )

- Get the URI of all nodes referring to this node. It returns the list of leaf nodes which, if they have value equal to the name of the current node, will prevent any renaming or deletion of this node.

Returns The URI list of nodes reffering to this node, or null if not defined.

11.4.13.7 public DmtData getDefault( )

- Get the default value of this node if any.

Returns The default value or null if not defined.

11.4.13.8 public String[] getDependentURIs( )

- Get the URI of all nodes referring to this node. It returns the list of URIs for leaf nodes whose value is changed when the current node is renamed, or set to null if the current node is deleted.

Returns The URI list of nodes reffering to this node, or null if not defined.

11.4.13.9 public String getDescription( )

- Get the explanation string associated with this node

Returns Node description string

11.4.13.10 public int getFormat( )

- Get the node's format, expressed in terms of type constants defined in DmtDataType. Note that the 'format' term is a legacy from OMA DM, it is more customary to think of this as 'type'.

Returns The format of the node.

11.4.13.11 public int getMax( )

- Get the maximum allowed value associated with this node.

Returns The allowed maximum. If the node's hasMax() returns false then Integer.MIN_VALUE is returned.

11.4.13.12 public int getMaxOccurrence( )

- Get the number of maximum occurrence of this type of nodes on the same level in the DMT.

Returns The maximum allowed occurrence of this node type

11.4.13.13 public String[] getMimeTypes( )

- Get the list of MIME types this node can hold.

Returns The list of allowed MIME types for this node or null if not defined.

11.4.13.14 public int getMin( )

- Get the minimum allowed value associated with this node.

Returns The allowed minimum. If the node's hasMin() returns false then Integer.MAX_VALUE is returned.

11.4.13.15 public String getReferredURI( )

- Get the URI of a node whose children's names are the only valid values for the current node. For example, let us assume that we have a node defining the connectivity profile for the browser application: ./DevDetail/Ext/Browser/Conn. The node is a leaf, containing the name of one of the profiles defined under ./DevDetail/Ext/DataProfiles. If invoked on the former, this method method will return the URI of the latter

Returns The URI of the referred node or null if not defined.

11.4.13.16 public String getRegExp( )

- Get the regular expression associated with this node if any. This method makes sense only in the case of chr nodes.

Returns The regular expression associated with this node or null if not defined, or if the node is not of type chr.

11.4.13.17 public DmtData[] getValidValues( )

- Return an array of DmtData objects if valid values are defined for the node, or null otherwise

Returns the valid values for this node, or null if not defined

11.4.13.18 public boolean hasMax( )

- Check whether the node's value has a maximum value associated with it

Returns true if the node's value has a maximum value, false if not or the node's format can not allow having a maximum

11.4.13.19 public boolean hasMin( )

- Check whether the node's value has a minimum value associated with it

Returns true if the node's value has a minimum value, false if not or the node's format can not allow having a minimum

11.4.13.20 public boolean isLeaf( )

- Check whether the node is a leaf node or an internal one

Returns true if the node is a leaf node

11.4.13.21 public boolean isPermanent( )

- Check whether the node is a permanent one. Note that a permanent node is not the same as a node where the DELETE operation is not allowed. Permanent nodes never can be deleted, whereas a non-deletable node can disappear in a recursive DELETE operation issued on one of its parents.

Returns true if the node is permanent

11.4.13.22 public boolean isZeroOccurrenceAllowed( )

- Check whether zero occurrence of this node is valid

Returns true if zero occurrence of this node is valid

11.4.14 public class DmtPrincipal

The representation of an entity having the role of manipulating the Device Management Tree.

11.4.14.1 public DmtPrincipal( )

- Create a DmtPrincipal instance. This empty constructor is based on an internal identity, such as bundle ID, code signer, subscription ID, carrier ID etc. The policies enforced for such an identity are expressed by the DmtPermission class. This constructor can be used by any application, and no special permission is defined to restrict its use. It is intended for use by applications not having management roles.

11.4.14.2 public DmtPrincipal(String name) throws DmtException

name The name associated with this principal

- Create a DmtPrincipal instance for the given named entity. This string-based constructor is used primarily by agents serving various DM protocols as well as administrative applications setting ACLs. In this case identities are external (not derivable from the device's environment), such as management server identities, determined during the authentication process. This constructor can be used only by the trusted code, and a special DmtPrincipalPermission class is defined to enforce this limitation. [TODO: check this] For example, the identity of the OMA DM server can be established during the handshake between the OMA DM agent and the server.

Throws `DmtException` –

11.4.14.3 public String getName( )

- Get the name associated with this principal.

Returns The name associated with this principal, or null if no name was specified at creation.

11.4.15 public interface DmtReadOnly

This interface collects basic DMT operations. The application programmers use these methods when they are interacting with a DmtSession which inherits from this interface.

11.4.15.1 public void close( ) throws DmtException

- Closes a session and makes the changes made to the DMT persistent. Persisting the changes works differently for exclusive and atomic lock. For the former all changes that were accepted are persisted. For the latter once an error is encountered, all successful changes are rolled back.

  This method can fail even if all operations were successful. This can happen due to some multi-node semantic constraints defined by a specific implementation. For example, node A can be required to always have children A.B, A.C and A.D. If this condition is broken when close() is executed, the method will fail, and throw an exception.

11.4.15.2 public String[] getChildNodeNames(String nodeUri) throws DmtException

nodeUri The URI of the node

- Get the list of children names of a node. The returned array contains the names - not the URIs - of the immediate children nodes of the given node.

Returns The list of children node names as a string array.

Throws `DmtException` –

11.4.15.3 public int getNodeSize(String nodeUri) throws DmtException

nodeUri The URI of the node

- Get the size of a node in bytes

Returns The size of the node

Throws `DmtException` –

11.4.15.4 public Date getNodeTimestamp(String nodeUri) throws DmtException

nodeUri The URI of the node

- Get the timestamp when the node was last modified

Returns The timestamp of the last modification

Throws `DmtException` –

11.4.15.5 public String getNodeTitle(String nodeUri) throws DmtException

nodeUri The URI of the node

- Get the title of a node

Returns The title of the node

Throws `DmtException` –

11.4.15.6 public String getNodeType(String nodeUri) throws DmtException

nodeUri The URI of the node

- Get the type of a node. The type of leaf node is the MIME type of the data it contains. The type of interior node is an URL pointing to a DDF document.

Returns The type of the node

Throws `DmtException` –

11.4.15.7 public DmtData getNodeValue(String nodeUri) throws DmtException

nodeUri The URI of the node to retrieve

- Get the data contained in a leaf node. Throws DmtException with the error code COMMAND_NOT_ALLOWED if issued on an interior node.

Returns The data of the leaf node

Throws `DmtException` –

11.4.15.8 public int getNodeVersion(String nodeUri) throws DmtException

nodeUri The URI of the node

- Get the version of a node. The version can not be set, it is calculated automatically by the device. It is incremented each time the value of the node is changed.

Returns The version of the node

Throws `DmtException` –

11.4.15.9 public boolean isNodeUri(String nodeUri)

nodeUri the URI to check

- Check whether the specified URI corresponds to a valid node in the DMT.

Returns true if the given node exists in the DMT

11.4.16 public interface DmtReadOnlyDataPlugIn extends DmtReadOnly

An implementation of this interface takes the responsibility over a non-modifiable subtree of the DMT. In an OSGi environment such implementations should be registered at the OSGi service registry specifying the list of root node URIs in a String array in the dataRootURIs registration parameter.

11.4.16.1 public DmtMetaNode getMetaNode(String nodeUri, DmtMetaNode generic) throws DmtException

nodeUri The URI of the node of which metadata information is queried

generic Metadata information supplied by the Dmt Admin

- Get metadata information about a given node. The plugin implementation may complement the metadata supplied by the Dmt Admin by its own, plugin specific metadata. This mechanism can be used, for example, when a data plugin is implemented on top of a data store or another API that have their own metadata, such as a relational database, in order to avoid metadata duplication and inconsistency.

Returns The complete metadata information

Throws `DmtException` –

11.4.16.2 public void open(String subtreeUri, DmtSession session) throws DmtException

subtreeUri The subtree which is locked in the current session

session The session from which this plugin instance is accessed

- This method is called to signal the start of a transaction when the first reference is made within a DmtSession to a node which is handled by this plugin. Although a read only plugin need not be concerned about transactionality, knowing the session from which it is accessed can be useful for example in the case of sending alerts.

Throws `DmtException` –

11.4.17 public interface DmtSession extends Dmt

DmtSession provides concurrent access to the DMT. All DMT manipulation commands for management applications are available on the DmtSession interface. The session is associated with a root node which limits the subtree in which the operations can be executed within this session. Most of the operations take a node URI as parameter, it can be either an absolute URI (starting with "./") or a URI relative to the root node of the session. If the URI specified does not correspond to a legitimate node in the tree an exception is thrown. The only exception is the isNodeUri() method which returns false in case of an invalid URI.

11.4.17.1 public static final int LOCK_TYPE_ATOMIC = 3

LOCK_TYPE_ATOMIC is an exclusive lock with transactional functionality.

11.4.17.2 public static final int LOCK_TYPE_AUTOMATIC = 2

LOCK_TYPE_AUTOMATIC starts as a shared lock, and is escalated to an exclusive one when an update operation is performed on the tree for the first time. This is the default lock type.

11.4.17.3 public static final int LOCK_TYPE_EXCLUSIVE = 1

LOCK_TYPE_EXCLUSIVE lock guarantees full access to the tree, but can not be shared with any other locks.

11.4.17.4 public static final int LOCK_TYPE_SHARED = 0

Sessions created with LOCK_TYPE_SHARED lock allows read-only access to the tree, but can be shared between multiple readers.

11.4.17.5 public void execute(String nodeUri, String data) throws DmtException

nodeUri the node on which the execute operation is issued

data the parameters to the execute operation. The format of the data string is described by the managed object definition.

- Executes a node. This corresponds to the EXEC operation in OMA DM. The semantics of an EXEC operation depend on the definition of the managed object on which it is issued.

Throws `DmtException` –

11.4.17.6 public int getLockType( )

- Gives the type of lock the session currently has, which might be different from the type it was created with.

Returns One of the LOCK_TYPE_... constants.

11.4.17.7 public DmtMetaNode getMetaNode(String nodeUri) throws DmtException

nodeUri the URI of the node

- Get the meta data which describes a given node. Meta data can be only inspected, it can not be changed.

Returns a DmtMetaNode which describes meta data information

Throws `DmtException` –

11.4.17.8 public DmtAcl getNodeAcl(String nodeUri) throws DmtException

nodeUri the URI of the node

- Gives the Access Control List associated with a given node.

Returns the Access Control List belonging to the node

Throws `DmtException` –

11.4.17.9 public String getPrincipal( )

- Gives the name of the principal on whose behalf the session was created. Local sessions do not have an associated principal, in this case null is returned.

Returns the identifier of the remote server that initiated the session, or null for local sessions

11.4.17.10 public String getRootUri( )

- Get the root URI associated with this session. Gives "." if the session was created without specifying a root.

Returns the root URI

11.4.17.11 public int getSessionId( )

- The unique identifier of the session. The ID is generated automatically, and it is guaranteed to be unique on a machine.

Returns the session identification number

11.4.17.12 public boolean isLeafNode(String nodeUri) throws DmtException

nodeUri the URI of the node

- Tells whether a node is a leaf or an interior node of the DMT.

Returns true if the given node is a leaf node

Throws `DmtException` –

11.4.17.13 public void setNodeAcl(String nodeUri, DmtAcl acl) throws DmtException

nodeUri the URI of the node

acl the Access Control List to be set on the node

- Set the Access Control List associated with a given node.

Throws `DmtException` –

12 Monitor Specification

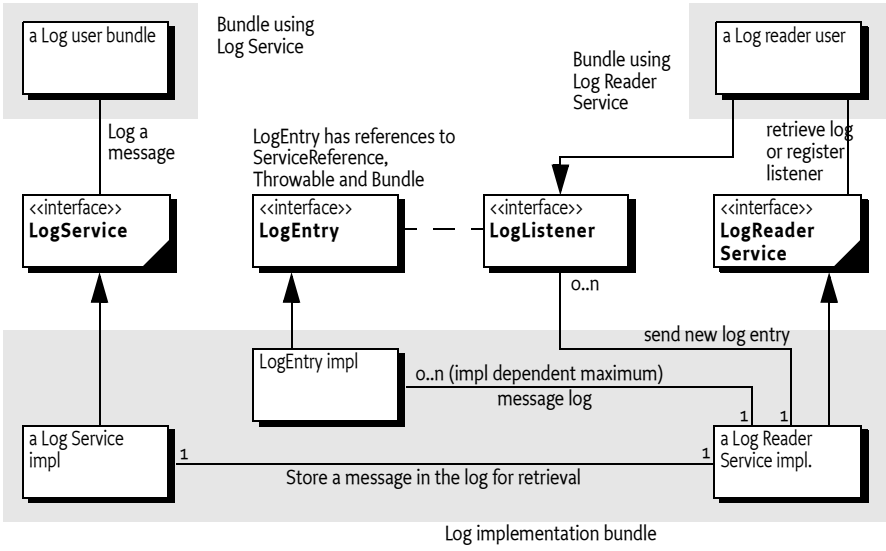
Version 1.0

12.1 Introduction

12.1.1 Entities

- *Application* – ....
- *Application Descrriptor* –

Figure 24 Log Service Class Diagram org.osgi.service.log package



12.2 The Meglet Base Class

12.3 Security

.

12.4 org.osgi.service.monitor

The OSGi Monitor Specification Version 1.0.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.montitor; specification-version=1.0
```

12.4.1 Summary

- KPI - A KPI object represents the value of a status variable taken with a certain collection method at a certain point of time. [p.134]
- Monitorable - A Monitorable can provide information about itself in the form of KPIs. [p.137]
- MonitorAdmin - A MonitorAdmin implementation handles KPI query requests and measurement job control requests. [p.138]
- MonitoringJob - A Monitoring Job is a scheduled update of a set of KPIs. [p.139]
- UpdateListener - The UpdateListener is used by Monitorable services to send notifications when a KPI value is changed. [p.140]

12.4.2 public class KPI

A KPI object represents the value of a status variable taken with a certain collection method at a certain point of time. The type of the KPI can be integer, float or String. Additionally Object type is supported which can hold any type supported by the OSGi Configuration Admin service. KPI objects are immutable.

12.4.2.1 public static final int CM_CC = 0

12.4.2.2 public static final int CM_DER = 1

12.4.2.3 public static final int CM_GAUGE = 2

12.4.2.4 public static final int CM_SI = 3

12.4.2.5 public static final int TYPE_FLOAT = 1

12.4.2.6 public static final int TYPE_INTEGER = 0

12.4.2.7 public static final int TYPE_OBJECT = 3

12.4.2.8 public static final int TYPE_STRING = 2

12.4.2.9 public KPI(String monitorableId, String id, String description, int cm, int data)

monitorableId The identifier of the monitorable service that this KPI belongs to

id The identifier of the KPI

description The human-readable description of the KPI

cm Collection method, should be one of the CM_ constants

data The integer value of the KPI

□ Constructor for a KPI of integer type.

Throws `IllegalArgumentException` – if any of the ID parameters contains invalid characters or if cm is not one of the collection method constants

`NullPointerException` – if any of the ID parameters is null

12.4.2.10 public KPI(String monitorableId, String id, String description, int cm, float data)

monitorableId The identifier of the monitorable service that this KPI belongs to

id The identifier of the KPI

description The human-readable description of the KPI

cm Collection method, should be one of the CM_ constants

data The float value of the KPI

□ Constructor for a KPI of float type.

Throws `IllegalArgumentException` – if any of the ID parameters contains invalid characters or if cm is not one of the collection method constants

`NullPointerException` – if any of the ID parameters is null

12.4.2.11 public KPI(String monitorableId, String id, String description, int cm, String data)

monitorableId The identifier of the monitorable service that this KPI belongs to

id The identifier of the KPI

description The human-readable description of the KPI

cm Collection method, should be one of the CM_ constants

data The string value of the KPI

□ Constructor for a KPI of String type.

Throws `IllegalArgumentException` – if any of the ID parameters contains invalid characters or if cm is not one of the collection method constants

`NullPointerException` – if any of the ID parameters is null

12.4.2.12 public KPI(String monitorableId, String id, String description, int cm, Object data)

monitorableId The identifier of the monitorable service that this KPI belongs to

id The identifier of the KPI

description The human-readable description of the KPI

cm Collection method, should be one of the CM_ constants

data The value of the KPI. The type of the object should be supported by the Configuration Admin

□ Constructor for a KPI of Object type.

Throws `IllegalArgumentException` – if any of the ID parameters contains invalid characters or if cm is not one of the collection method constants

`NullPointerException` – if any of the ID parameters is null

12.4.2.13 public boolean equals(Object obj)

obj the object to compare with this KPI

□ Compares the specified object with this KPI. Two KPI objects are considered equal if their full path, description (if any), collection method and type are identical, and the data (selected by their type) is equal.

Returns true if the argument represents the same KPI as this object

12.4.2.14 public int getCollectionMethod( )

- Returns the collection method of this KPI. See section 3.3 b) in [ETSI TS 132 403]

Returns one of the CM_ constants

12.4.2.15 public String getDescription( )

- Returns a human readable description of this KPI. This can be used by management systems on their GUI. Null return value is allowed.

Returns the human readable description of this KPI or null if it is not set

12.4.2.16 public float getFloat( ) throws IllegalStateException

- Returns the KPI value if its type is float.

Returns the KPI value as a float

Throws `IllegalStateException` – if the type of this KPI is not float

12.4.2.17 public String getID( )

- Returns the name of this KPI. A KPI name is unique within the scope of a Monitorable. A KPI name must not contain the Reserved characters described in 2.2 of RFC-2396 (URI Generic Syntax).

Returns the name of this KPI

12.4.2.18 public int getInteger( ) throws IllegalStateException

- Returns the KPI value if its type is integer.

Returns the KPI value as an integer

Throws `IllegalStateException` – if the type of this KPI is not integer

12.4.2.19 public Object getObject( ) throws IllegalStateException

- Returns the KPI value if its type is Object. All types supported by the OSGi Configuration Admin service can be used. Exact type information can be queried using getClass().getName().

Returns the KPI value as an Object

Throws `IllegalStateException` – if the type of this KPI is not Object

12.4.2.20 public String getPath( )

- Returns the path (long name) of this KPI. The path of the KPI is created from the ID of the Monitorable service it belongs to and the ID of the KPI in the following form: [Monitorable_id]/[kpi_id].

Returns the name of this KPI

12.4.2.21 public String getString( ) throws IllegalStateException

- Returns the KPI value if its type is String.

Returns the KPI value as a string

Throws `IllegalStateException` – if the type of the KPI is not String

12.4.2.22 public Date getTimeStamp( )

- Returns the time when the KPI value was queried. The KPI's value is set when the getKpi() or getKpis() methods are called on the Monitorable object.

Returns the time when the KPI value was queried

12.4.2.23 public int getType( )

- Returns information on the data type of this KPI.

Returns one value of the set of type constants

12.4.2.24 public int hashCode( )

- Returns the hash code value for this KPI. The hash code is calculated based on the full path, description (if any), collection method and value data of the KPI.

12.4.2.25 public String toString( )

- Returns a string representation of this KPI. The returned string contains the full path, the collection method, the exact time of creation, the description (if any), the type and the value of the KPI in the following format: KPI(‹path›, ‹cm›, [‹description›,]‹timestamp›, ‹type›, ‹value›)

Returns the string representation of this KPI

12.4.3 public interface Monitorable

A Monitorable can provide information about itself in the form of KPIs. Instances of this interface should register themselves at the OSGi Service registry. The MonitorAdmin listens to the registration of Monitorable services, and makes the information they provide available also through the Device Management Tree (DMT). The monitorable service is identified by its PID string which must not contain the Reserved characters described in 2.2 of RFC-2396 (URI Generic Syntax).

A Monitorable may optionally support sending notifications when the status of its KPIs change.

12.4.3.1 public KPI getKpi(String id) throws IllegalArgumentException

id the identifier of the KPI. The method returns the same KPI regardless of whether the short or long ID is used.

- Returns the KPI object addressed by its identifier. The KPI will hold the value taken at the time of this method call.

Returns the KPI object

Throws `IllegalArgumentException` – if the path is invalid or points to a non existing KPI

12.4.3.2 public String[] getKpiNames( )

- Returns the list of KPI identifiers published by this Monitorable. A KPI name is unique within the scope of a Monitorable. The array contains the elements in no particular order.

Returns the name of KPIs published by this object

12.4.3.3 public String[] getKpiPaths( )

- Returns the list of long KPI names published by this Monitorable. This name is the combination of the ID of the Monitorable and the name of the KPI in the following format: [Monitorable_ID]/[KPI_name], for example MyApp/QueueSize. This name is guaranteed to be unique on the service platform and it is used in a Monitoring's list of observed KPIs. The array contains the elements in no particular order.

Returns the 'fully qualified' name of KPIs published by this object

12.4.3.4 public KPI[] getKpis( )

- Returns all the KPI objects published by this Monitorable instance. The KPIs will hold the values taken at the time of this method call. The array contains the elements in no particular order.

Returns the KPI objects published by this Monitorable instance

12.4.3.5 public boolean notifiesOnChange(String id) throws IllegalArgumentException

id the identifier of the KPI. The method works the same way regardless of whether the short or long ID is used.

- Tells whether the KPI provider is able to send instant notifications when the given KPI changes. If the Monitorable supports sending change updates it must notify the UpdateListener when the value of the KPI changes. The Monitorable finds the UpdateListener service through the Service Registry.

Returns true if the Monitorable can send notification when the given KPI chages, false otherwise.

Throws `IllegalArgumentException` – if the path is invalid or points to a non existing KPI

12.4.3.6 public boolean resetKpi(String id) throws IllegalArgumentException

id the identifier of the KPI. The method works the same way regardless of whether the short or long ID is used.

- Issues a request to reset a given KPI. Depending on the semantics of the KPI this call may or may not succeed: it makes sense to reset a counter to its starting value, but e.g. a KPI of type String might not have a meaningful default value. Note that for numeric KPIs the starting value may not necessarily be 0. Resetting a KPI triggers a monitor event.

Returns true if the Monitorable could successfully reset the given KPI, false otherwise

Throws `IllegalArgumentException` – if the path is invalid or points to a non existing KPI

12.4.4 public interface MonitorAdmin

A MonitorAdmin implementation handles KPI query requests and measurement job control requests.

12.4.4.1 public KPI getKPI(String path) throws IllegalArgumentException

path the full path of the KPI in [Monitorable_ID]/[KPI_ID] format

- Returns a KPI addressed by its ID in [Monitorable_ID]/[KPI_ID] format. This is a convenience feature for the cases when the full path of the KPI is known. KPIs can also be obtained by querying the list of Monitorable services from the service registry and then querying the list of KPI names from the Monitorable services.

Returns the KPI object

Throws `IllegalArgumentException` – if the path is invalid or points to a non existing KPI

12.4.4.2 public MonitoringJob[] getRunningJobs( )

- Returns the list of currently running Monitoring Jobs.

Returns the list of running jobs

12.4.4.3 public MonitoringJob startJob(String initiator, String[] kpis, int schedule, int count) throws IllegalArgumentException

initiator the identifier of the entity that initiated the job

kpis list of KPIs to be monitored. The KPI names must be given in [Monitorable_PID]/[KPI_ID] format

schedule The time in seconds between two measurements. The value 0 means that instant notification on KPI updates is requested. For values greater than 0 the first measurement will be taken when the timer expires for the first time, not when this method is called.

count For time based monitoring jobs, this should be the number of measurements to be taken, or 0 if the measurement must run infinitely. For change based monitoring (if schedule is 0), this must be a positive interger that specifies the number of changes that must happen before a new notification is sent.

- Starts a Monitoring Job with the parameters provided. If the list of KPI names contains a non existing KPI or the schedule or count parameters are invalid then IllegalArgumentException is thrown. The initiator string is used in the listener.id field of all events triggered by the job, to allow filtering the events based on the initiator.

Returns the successfully started job object

Throws `IllegalArgumentException` –

12.4.5 public interface MonitoringJob

A Monitoring Job is a scheduled update of a set of KPIs. The job is a data structure that holds a non-empty list of KPI names, an identification of the initiator of the job, and the sampling parameters. There are two kinds of monitoring jobs: time based and change based. Time based jobs take samples of all KPIs with a specified frequency. The number of samples to be taken before the job finishes may be specified. Change based jobs are only interested in the changes of the monitored KPIs. In this case, the number of changes that must take place between two notifications can be specified.

The job can be started on the MonitorAdmin interface. Running the job (querying the KPIs, listening to changes, and sending out notifications on updates) is the task of the MonitorAdmin implementation.

Whether a monitoring job keeps track dynamically of the KPIs it monitors is not specified. This means that if we monitor a KPI of a Monitorable service which disappears and later reappears then it is implementation specific whether we still receive updates of the KPI changes or not.

12.4.5.1 public String getInitiator( )

- Returns the identitifier of the principal who initiated the job. This is set at the time when startJob() is called at the MonitorAdmin interface. This string holds the ServerID if the operation was initiated from a remote manager, or an arbitrary ID of the initiator entity in the local case (used for addressing notification events).

Returns the ID of the initiator

12.4.5.2 public String[] getKpiNames( )

- Returns the list of KPI names which is the target of this measurement job. For time based jobs, the MonitorAdmin will iterate through this list and query all KPIs when its timer set by the job's frequency rate expires.

Returns the target list of the measurement job in [Monitorable_ID]/[KPI_ID] format

12.4.5.3 public int getReportCount( )

- Returns the number times MonitorAdmin will query the KPIs (for time based jobs), or the number of changes of a KPI between notifications (for change based jobs). Time based jobs with non-zero report count will take getReportCount()∗getSchedule() time to finish. Time based jobs with 0 report count and change based jobs do not stop automatically, but all jobs can be stopped with the stop method.

Returns the number of measurements to be taken, or the number of changes between notifications

12.4.5.4 public long getSchedule( )

- Returns the delay (in seconds) between two samples. If this call returns N (greater than 0) then the MonitorAdmin queries each KPI that belongs to this job every N seconds. The value 0 means that instant notification on changes is requested (at every nth change of the value, as specified by the report count parameter).

Returns the delay (in seconds) between samples, or 0 for change based jobs

12.4.5.5 public boolean isLocal( )

- Returns whether the job was started locally or remotely.

Returns true if the job was started from the local device, false if the job was initiated from a remote management server through the device management tree

12.4.5.6 public void stop( )

- Stops a Monitoring Job. Note that a time based job can also stop automatically if the specified number of samples have been taken.

12.4.6 public interface UpdateListener

The UpdateListener is used by Monitorable services to send notifications when a KPI value is changed. The UpdateListener should register itself as a service at the OSGi Service Registry. This interface is implemented by the Monitor Admin component.

12.4.6.1 public void updated(KPI kpi)

kpi the KPI which has changed

□ Callback for notification of a KPI change.

13 MEG Specification

Version 1.0

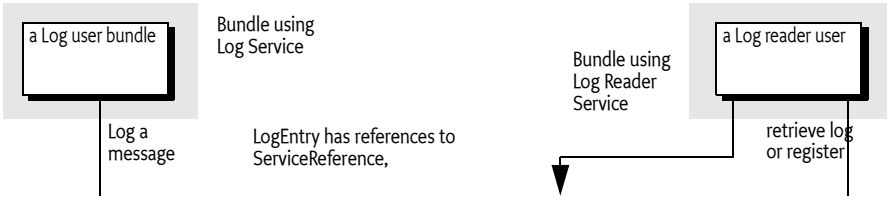
13.1 Introduction

Diverse classes specific for MEG. The name is wrong. DMTPermission does not belong here. IMSI and IMEI seem to indicate a GSM class?

13.1.1 Entities

- *Application* – ....
- *Application Descrriptor* –

Figure 25 *Log Service Class Diagram org.osgi.service.log package*



13.2 The MEG Classses

13.3 Security

13.4 org.osgi.meg

13.4.1 Summary

- DMTPermission - DMTPermission controls access to management objects in the Device Management Tree (DMT). [p.144]
- IMEI - Class representing an IMEI condition. [p.145]
- IMSI - Class representing an IMSI condition. [p.146]
- MEGMgmtPermission - MEGMgmtPermission controls access to MEG management framework functions. [p.146]
- TransferCost - Class representing abstract transfer cost values. [p.148]
- UserPrompt - Class representing a user prompt condition. [p.149]

13.4.2 public class DMTPermission extends Permission

DMTPermission controls access to management objects in the Device Management Tree (DMT). It is intended to control local access to the DMT. DMTPermission target string identifies the management object URI and the action field lists the OMA DM commands that are permitted on the management object. Example:

```
DMTPermission(". /OSGi/bundles", "Add,Replace,Get");
```

This means that owner of this permission can execute Add, Replace and Get commands on the ./OSGi/bundles management object. It is possible to use wildcards in both the target and the actions field. Wildcard in the target field means that the owner of the permission can access children nodes of the target node. Example

```
DMTPermission(". /OSGi/bundles/*", "Get");
```

This means that owner of this permission has Get access on every child node of ./OSGi/bundles. If wildcard is present in the actions field, all legal OMA DM commands are allowed on the designated nodes(s) by the owner of the permission.

13.4.2.1 public DMTPermission(String dmturi, String actions)

dmturi URI of the management object (or subtree).

actions OMA DM actions allowed.

- Creates a new FilePermission object for the specified DMT URI with the specified actions.

13.4.2.2 public boolean equals(Object obj)

- Checks two DMTPermission objects for equality. Two DMTPermissions are equal if they have the same target and action strings.

Returns true if the two objects are equal.

13.4.2.3 public String getActions( )

- Returns the String representation of the action list.

Returns Action list for this permission object.

13.4.2.4 public int hashCode( )

- Returns hash code for this permission object. If two DMTPermission objects are equal according to the equals method, then calling the hashCode method on each of the two DMTPermission objects must produce the same integer result.

Returns hash code for this permission object.

13.4.2.5 public boolean implies(Permission p)

p Permission to check.

- Checks if this DMTPermission object "implies" the specified permission.

Returns true if this DMTPermission object implies the specified permission.

13.4.2.6 public PermissionCollection newPermissionCollection( )

- Returns a new PermissionCollection object for storing DMTPermission objects.

Returns the new PermissionCollection.

13.4.3 public class IMEI implements Condition

Class representing an IMEI condition. Instances of this class contain a string value that is matched against the IMEI of the device.

13.4.3.1 public IMEI(Bundle bundle, String imei)

bundle ignored

imei The IMEI value of the device.

- Creates an IMEI object.

13.4.3.2 public boolean isEvaluated( )

- Checks whether the condition is evaluated.

Returns always true

13.4.3.3 public boolean isMutable( )

- check whether the condition can change.

Returns always false

13.4.3.4 public boolean isSatisfied( )

- Checks whether the condition is satisfied. The IMEI of the object instance is compared against the IMEI of the device.

Returns true if the IMEI value matches.

13.4.3.5 public boolean isSatisfied(Condition[] conds, Dictionary context)

conds an array, containing only IMEI conditions

context ignored

- Checks for an array of IMEI conditions if they all match this device

Returns true if all conditions match

13.4.4 public class IMSI implements Condition

Class representing an IMSI condition. Instances of this class contain a string value that is matched against the IMSI of the subscriber.

13.4.4.1 public IMSI(Bundle bundle, String imsi)

bundle ignored

imsi The IMSI value of the subscriber.

- Creates an IMSI object.

13.4.4.2 public boolean isEvaluated( )

- Checks whether the condition is evaluated

Returns always true

13.4.4.3 public boolean isMutable( )

- Checks whether the condition can change

Returns always false

13.4.4.4 public boolean isSatisfied( )

- Checks whether the condition is true. The IMSI of the object instance is compared against the IMSI of the device's user.

Returns true if the IMSI value match.

13.4.4.5 public boolean isSatisfied(Condition[] conds, Dictionary context)

conds an array, containing only IMSI conditions

context ignored

- Checks whether an array of IMSI conditions match

Returns true, if they all match

13.4.5 public class MEGMgmtPermission extends Permission

MEGMgmtPermission controls access to MEG management framework functions. This permission controls MEG-only functions, framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) MEGMgmtPermission has two parameters: a target specifier and an action string.

```
MEGMgmtPermission("<filter>", "inventory")
```

Holder of this permissions can access the inventory information of the bundles selected by the ‹filter› string. The ‹filter› string is defined identically to the RFC 73 filter. The filter selects the bundles on which the holder of the permission can acquire detailed inventory information.

```
MEGMgmtPermission("<monitorable ID>/<KPI ID>", "KPIpublish")
```

Holder of this permission is allowed to publish a KPI under monitorable ID/KPI ID. Wildcard (*) is allowed to represent both monitorable ID and KPI ID.

```
MEGMgmtPermission("<filter>", "KPIperiodical")
```

Holder of this permission is allowed to launch a periodical KPI update jobs. ‹Filter› is an OSGi filter as defined by org.osgi.framework.Filter. Fields of the filter can be period and reports. Period is the minimum sampling time between samples (in milliseconds) while reports is the maximum length of the monitoring job in minutes. The addressee of the periodical KPI sampling results is always the initiator of the KPI update job, hence this information is implied by the entity which holds the permission.

```
MEGMgmtPermission("<filter>", "integrity")
```

Holder of this permissions can launch integrity check on the bundles selected by the ‹filter› string. The ‹filter› string is defined identically to the RFC 73 filter.

13.4.5.1 public MEGMgmtPermission(String target, String action)

target Target string.

action Action string.

- Creates a new MEGMgmtPermission for the given target and action.

13.4.5.2 public boolean equals(Object obj)

- Checks two MEGMgmtPermission objects for equality. Two MEGMgmtPermission objects are equal if their target and action strings are equal.

Returns true if the two objects are equal.

13.4.5.3 public String getActions( )

- Returns the String representation of the action list.

Returns Action list of this permission instance. This is a comma-separated list that reflects the action parameter of the constructor.

13.4.5.4 public int hashCode( )

- Returns hash code for this permission object.

Returns hash code for this permission object.

13.4.5.5 public boolean implies(Permission p)

p Permission to check.

- Checks if this MEGMgmtPermission would imply the parameter permission.

Returns true if this MEGMgmtPermission object implies the specified permission.

13.4.5.6 public PermissionCollection newPermissionCollection( )

- Returns a new PermissionCollection object for storing MEGMgmtPermission objects.

Returns the new PermissionCollection.

13.4.6 public class TransferCost implements Condition

Class representing abstract transfer cost values. Implementation assigns concrete bearers to abstract transfer costs.
There is a static setTransferCost() method that sets the current transfer cost limit in a thread local variable. The TransferCost condition will check this, and will fail if the cost is greater. The setTransferCost() and resetTransferCost() methods behave stack-like, putting and removing costs from a stack. The limit is always the value on the top. This way nested permission checks can be implemented. It is recommended that resetTransferCost() is always put in a finally block, to ensure proper stacking behavior.

13.4.6.1 public TransferCost(Bundle bundle, String cost)

bundle ignored

cost The abstract limit cost. Possible values are “LOW”,”MEDIUM” and “HIGH”.

- Creates a TransferCost object. This constructor is intended to be called when Permission Admin initializes the object.

13.4.6.2 public boolean isEvaluated( )

- Checks whether the condition is evaluated.

Returns always false

13.4.6.3 public boolean isMutable( )

- check whether the condition can change.

Returns always true

13.4.6.4 public boolean isSatisfied( )

- Checks whether the condition is satisfied. The limit cost represented by this object instance is compared against the cost in the thread context.

Returns true if the cost in this condition is greater or equals the cost limit set in thread context

13.4.6.5 public boolean isSatisfied(Condition[] conds, Dictionary context)

conds an array, containing only TransferCost conditions

context ignored

- Checks for an array of TransferCost conditions if they all match

Returns true if all transfer costs are below limit

13.4.6.6 public static void resetTransferCost( )

- Resets the transfer cost to the previous value.

13.4.6.7 public static void setTransferCost(int cost)

cost the upper limit of transfer cost

- Sets a thread-local transfer cost limit. All isSatisfied() method calls in this thread will check for this limit. The caller MUST call resetTransferCost(), after the permission checks are done. If this function is not called, the default behavior is 'no limit'.

13.4.7 public class UserPrompt implements Condition

Class representing a user prompt condition. Instances of this class hold two values: a prompt string that is to be displayed to the user and the permission level string according to MIDP2.0 (oneshot, session, blanket).

13.4.7.1 public static final String BLANKET = "blanket"

13.4.7.2 public static final String ONESHOT = "oneshot"

13.4.7.3 public UserPrompt(Bundle bundle, String prompt, String level)

bundle the bundle to ask about

prompt The message to display to the user.

level The permission level. This must be ONESHOT, SESSION or BLANKET.

- Creates an UserPrompt object with the given prompt string and permission level. There is one interaction mode with this permission and that is the default as well.

13.4.7.4 public UserPrompt(Bundle bundle, String prompt, String deflevel, String level1, String level2, String level3)

bundle the bundle to ask about

prompt The message to display to the user.

deflevel The default level. If it is "X" then there's no default level, if it is "0" then level1 is the default level.

level1 1st permission level. This must be ONESHOT, SESSION, BLANKET or X if this field does not convey a permission level.

level2 2nd permission level. This must be ONESHOT, SESSION, BLANKET or X if this field does not convey a permission level.

level3 3rd permission level. This must be ONESHOT, SESSION, BLANKET or X if this field does not convey a permission level.

- Creates an UserPrompt object with the given prompt string and permission levels. The user should be given choice as to what level of permission is given. Thus, the lifetime of the permission is controlled by the user.

13.4.7.5 public static void clearDatabase( )

13.4.7.6 public boolean isEvaluated( )

- Checks whether the condition is evaluated. Depending on the permission level, the function returns the following values.
  - ONESHOT - isSatisfied always returns false. The condition reevaluated each time it is checked.
  - SESSION - isSatisfied returns false until isEvaluated() returns true (the user accepts the prompt for this execution of the bundle). Then isSatisfied() returns true until the bundle is stopped.
  - BLANKET - isSatisfied returns false until isEvaluated() returns true (the user accepts the prompt for the lifetime of the bundle). Then isSatisfied() returns true until the bundle is uninstalled.

Returns true if the condition is satisfied.

13.4.7.7 public boolean isMutable( )

- Checks whether the condition may change in the future.

Returns false, if it is already evaluated and is a BLANKET condition true otherwise

13.4.7.8 public boolean isSatisfied( )

- Checks whether the condition is satisfied. Displays the prompt string to the user and returns true if the user presses “accept”. Depending on the amount of levels the condition is assigned to, the prompt may have multiple “accept” buttons and one of them can be selected by default (see deflevel parameter at UserPrompt constructor). It must be always possible to deny the permission (e.g. by a separate “deny” button).

Returns true if the user accepts the prompt (or accepts any prompt in case there are multiple permission levels).

13.4.7.9 public boolean isSatisfied(Condition[] conds, Dictionary context)

conds the array containing the UserPrompt conditions to evaluate

context storage area for one-shot evaluation

- Checks an array of UserPrompt conditions

Returns true, if all conditions are satisfied

13.4.7.10 public static void setPromptHandler(Object handler)

14 XML Parser Service Specification

Version 1.0

14.1 Introduction

The Extensible Markup Language (XML) has become a popular method of describing data. As more bundles use XML to describe their data, a common XML Parser becomes necessary in an embedded environment in order to reduce the need for space. Not all XML Parsers are equivalent in function, however, and not all bundles have the same requirements on an XML parser.

This problem was addressed in the Java API for XML Processing, see [18] *JAXP* for Java 2 Standard Edition and Enterprise Edition. This specification addresses how the classes defined in JAXP can be used in an OSGi Service Platform. It defines how:

- Implementations of XML parsers can become available to other bundles
- Bundles can find a suitable parser
- A standard parser in a JAR can be transformed to a bundle

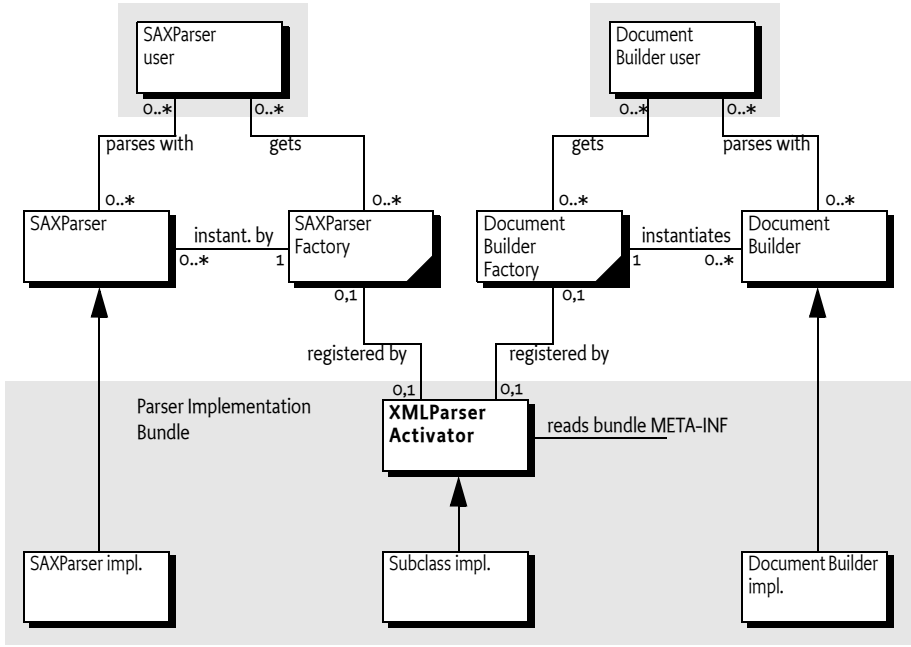
14.1.1 Essentials

- *Standards* – Leverage existing standards in Java based XML parsing: JAXP, SAX and DOM
- *Unmodified JAXP code* – Run unmodified JAXP code
- *Simple* – It should be easy to provide a SAX or DOM parser as well as easy to find a matching parser
- *Multiple* – It should be possible to have multiple implementations of parsers available
- *Extendable* – It is likely that parsers will be extended in the future with more functionality

14.1.2 Entities

- *XMLParserActivator* – A utility class that registers a parser factory from declarative information in the Manifest file.
- *SAXParserFactory* – A class that can create an instance of a `SAXParser` class.
- *DocumentBuilderFactory* – A class that can create an instance of a `DocumentBuilder` class.
- *SAXParser* – A parser, instantiated by a `SaxParserFactory` object, that parses according to the SAX specifications.
- *DocumentBuilder* – A parser, instantiated by a `DocumentBuilderFactory`, that parses according to the DOM specifications.

Figure 26 XML Parsing diagram



14.1.3 Operations

A bundle containing a SAX or DOM parser is started. This bundle registers a SAXParserFactory and/or a DocumentBuilderFactory service object with the Framework. Service registration properties describe the features of the parsers to other bundles. A bundle that needs an XML parser will get a SAXParserFactory or DocumentBuilderFactory service object from the Framework service registry. This object is then used to instantiate the requested parsers according to their specifications.

14.2 JAXP

XML has become very popular in the last few years because it allows the interchange of complex information between different parties. Though only a single XML standard exists, there are multiple APIs to XML parsers, primarily of two types:

- The Simple API for XML (SAX1 and SAX2)
- Based on the Document Object Model (DOM 1 and 2)

Both standards, however, define an abstract API that can be implemented by different vendors.

A given XML Parser implementation may support either or both of these parser types by implementing the org.w3c.dom and/or org.xml.sax packages. In addition, parsers have characteristics such as whether they are validating or non-validating parsers and whether or not they are name-space aware.

An application which uses a specific XML Parser must code to that specific parser and become coupled to that specific implementation. If the parser has implemented [18] *JAXP*, however, the application developer can code against SAX or DOM and let the runtime environment decide which parser implementation is used.

JAXP uses the concept of a *factory*. A factory object is an object that abstracts the creation of another object. JAXP defines a DocumentBuilderFactory and a SAXParserFactory class for this purpose.

JAXP is implemented in the javax.xml.parsers package and provides an abstraction layer between an application and a specific XML Parser implementation. Using JAXP, applications can choose to use any JAXP compliant parser without changing any code, simply by changing a System property which specifies the SAX- and DOM factory class names.

In JAXP, the default factory is obtained with a static method in the SAXParserFactory or DocumentBuilderFactory class. This method will inspect the associated System property and create a new instance of that class.

14.3 XML Parser service

The current specification of JAXP has the limitation that only one of each type of parser factories can be registered. This specification specifies how multiple SAXParserFactory objects and DocumentBuilderFactory objects can be made available to bundles simultaneously.

Providers of parsers should register a JAXP factory object with the OSGi service registry under the factory class name. Service properties are used to describe whether the parser:

- Is validating
- Is name-space aware
- Has additional features

With this functionality, bundles can query the OSGi service registry for parsers supporting the specific functionality that they require.

14.4 Properties

Parsers must be registered with a number of properties that qualify the service. In this specification, the following properties are specified:

- PARSER_NAMESPACEAWARE – The registered parser is aware of name-spaces. Name-spaces allow an XML document to consist of independently developed DTDs. In an XML document, they are recognized by the xmlns attribute and names prefixed with an abbreviated name-space identifier, like: <xsl:if ...>. The type is a Boolean object that must be true when the parser supports name-spaces. All other values, or the absence of the property, indicate that the parser does not implement name-spaces.
- PARSER_VALIDATING – The registered parser can read the DTD and can validate the XML accordingly. The type is a Boolean object that must

true when the parser is validating. All other values, or the absence of the property, indicate that the parser does not validate.

14.5 Getting a Parser Factory

Getting a parser factory requires a bundle to get the appropriate factory from the service registry. In a simple case in which a non-validating, non-name-space aware parser would suffice, it is best to use getServiceReference(String).

```
DocumentBuilder getParser(BundleContext context)
   throws Exception {
   ServiceReference ref = context.getServiceReference(
      DocumentBuilderFactory.class.getName() );
   if ( ref == null )
      return null;
   DocumentBuilderFactory factory =
      (DocumentBuilderFactory) context.getService(ref);
   return factory.newDocumentBuilder();
}
```

In a more demanding case, the filtered version allows the bundle to select a parser that is validating and name-space aware:

```
SAXParser getParser(BundleContext context)
   throws Exception {
   ServiceReference refs[] = context.getServiceReferences(
      SAXParserFactory.class.getName(),
         "(&(parser.namespaceAware=true)"
      +  "(parser.validating=true))" );
   if ( refs == null )
      return null;
   SAXParserFactory factory =
      (SAXParserFactory) context.getService(refs[0]);
   return factory.newSAXParser();
}
```

14.6 Adapting a JAXP Parser to OSGi

If an XML Parser supports JAXP, then it can be converted to an OSGi aware bundle by adding a BundleActivator class which registers an XML Parser Service. The utility org.osgi.util.xml.XMLParserActivator class provides this function and can be added (copied, not referenced) to any XML Parser bundle, or it can be extended and customized if desired.

14.6.1 JAR Based Services

Its functionality is based on the definition of the [19] *JAR File specification, services directory*. This specification defines a concept for service providers. A JAR file can contain an implementation of an abstractly defined service. The class (or classes) implementing the service are designated from a file in the META-INF/services directory. The name of this file is the same as the abstract service class.

The content of the UTF-8 encoded file is a list of class names separated by new lines. White space is ignored and the number sign ('#' or \u0023) is the comment character.

JAXP uses this service provider mechanism. It is therefore likely that vendors will place these service files in the META-INF/services directory.

14.6.2 XMLParserActivator

To support this mechanism, the XML Parser service provides a utility class that should be normally delivered with the OSGi Service Platform implementation. This class is a Bundle Activator and must start when the bundle is started. This class is copied into the parser bundle, and *not* imported.

The start method of the utlity BundleActivator class will look in the META-INF/services service provider directory for the files javax.xml.parsers.SAXParserFactory (SAXFACTORYNAME) or javax.xml.parsers.DocumentBuilderFactory (DOMFACTORYNAME). The full path name is specified in the constants SAXCLASSFILE and DOMCLASSFILE respectively.

If either of these files exist, the utility BundleActivator class will parse the contents according to the specification. A service provider file can contain multiple class names. Each name is read and a new instance is created. The following example shows the possible content of such a file:

```
# ACME example SAXParserFactory file
com.acme.saxparser.SAXParserFast        # Fast
com.acme.saxparser.SAXParserValidating  # Validates
```

Both the javax.xml.parsers.SAXParserFactory and the javax.xml.parsers.DocumentBuilderFactory provide methods that describe the features of the parsers they can create. The XMLParserActivator activator will use these methods to set the values of the properties, as defined in *Properties* on page 153, that describe the instances.

14.6.3 Adapting an Existing JAXP Compatible Parser

To incorporate this bundle activator into a XML Parser Bundle, do the following:

- If SAX parsing is supported, create a /META-INF/services/javax.xml.parsers.SAXParserFactory resource file containing the class names of the SAXParserFactory classes.
- If DOM parsing is supported, create a /META-INF/services/javax.xml.parsers.DocumentBuilderFactory file containing the fully qualified class names of the DocumentBuilderFactory classes.

- Create manifest file which imports the packages org.w3c.dom, org.xml.sax, and javax.xml.parsers.
- Add a Bundle-Activator header to the manifest pointing to the XMLParserActivator, the sub-class that was created, or a fully custom one.
- If the parsers support attributes, properties, or features that should be registered as properties so they can be searched, extend the XMLParserActivator class and override setSAXProperties(javax.xml.parsers.SAXParserFactory,Hashtable) and setDOMProperties(javax.xml.parsers.DocumentBuilderFactory,Hashtable).
- Ensure that custom properties are put into the Hashtable object. JAXP does not provide a way for XMLParserActivator to query the parser to find out what properties were added.
- Bundles that extend the XMLParserActivator class must call the original methods via super to correctly initialize the XML Parser Service properties.
- Compile this class into the bundle.
- Install the new XML Parser Service bundle.
- Ensure that the org.osgi.util.xml.XMLParserActivator class is is contained in the bundle.

14.7 Usage of JAXP

A single bundle should export the JAXP, SAX, and DOM APIs. The version of contained packages must be appropriately labeled. JAXP 1.1 or later is required which references SAX 2 and DOM 2. See [18] *JAXP* for the exact version dependencies.

This specification is related to related packages as defined in the JAXP 1.1 document. Table 9 contains the expected minimum versions.

Package	Minimum Version
javax.xml.parsers	1.1
org.xml.sax	2.0
org.xml.sax.helpers	2.0
org.xsml.sax.ext	1.0
org.w3c.dom	2.0

Table 9 *JAXP 1.1 minimum package versions*

The Xerces project from the Apache group, [20] *Xerces 2 Java Parser*, contains a number libraries that implement the necessary APIs. These libraries can be wrapped in a bundle to provide the relevant packages.

14.8 Security

A centralized XML parser is likely to see sensitive information from other bundles. Provisioning an XML parser should therefore be limited to trusted bundles. This security can be achieved by providing ServicePermission[REGISTER,javax.xml.parsers.DocumentBuilderFactory | javax.xml.parsers.SAXFactory] to only trusted bundles.

Using an XML parser is a common function, and ServicePermission[GET, javax.xml.parsers.DOMParserFactory | javax.xml.parsers.SAXFactory] should not be restricted.

The XML parser bundle will need FilePermission[<<ALL FILES>>,READ] for parsing of files because it is not known beforehand where those files will be located. This requirement further implies that the XML parser is a system bundle that must be fully trusted.

14.9 org.osgi.util.xml

The OSGi XML Parser service Package. Specification Version 1.0.

14.9.1 public class XMLParserActivator implements BundleActivator , ServiceFactory

A BundleActivator class that allows any JAXP compliant XML Parser to register itself as an OSGi parser service. Multiple JAXP compliant parsers can concurrently register by using this BundleActivator class. Bundles who wish to use an XML parser can then use the framework's service registry to locate available XML Parsers with the desired characteristics such as validating and namespace-aware.

The services that this bundle activator enables a bundle to provide are:

- javax.xml.parsers.SAXParserFactory(SAXFACTORYNAME[p.158])
- javax.xml.parsers.DocumentBuilderFactory(DOMFACTORYNAME[p.158])

The algorithm to find the implementations of the abstract parsers is derived from the JAR file specifications, specifically the Services API.

An XMLParserActivator assumes that it can find the class file names of the factory classes in the following files:

- /META-INF/services/javax.xml.parsers.SAXParserFactory is a file contained in a jar available to the runtime which contains the implementation class name(s) of the SAXParserFactory.
- /META-INF/services/javax.xml.parsers.DocumentBuilderFactory is a file contained in a jar available to the runtime which contains the implementation class name(s) of the DocumentBuilderFactory

If either of the files does not exist, XMLParserActivator assumes that the parser does not support that parser type.

XMLParserActivator attempts to instantiate both the SAXParserFactory and the DocumentBuilderFactory. It registers each factory with the framework along with service properties:

- PARSER_VALIDATING[p.158] - indicates if this factory supports validating parsers. It's value is a Boolean.
- PARSER_NAMESPACEAWARE[p.158] - indicates if this factory supports namespace aware parsers It's value is a Boolean.

Individual parser implementations may have additional features, properties, or attributes which could be used to select a parser with a filter. These can be added by extending this class and overriding the setSAXProperties and setDOMProperties methods.

14.9.1.1 public static final String DOMCLASSFILE = "/META-INF/services/javax.xml.parsers.DocumentBuilderFactory"

Fully qualified path name of DOM Parser Factory Class Name file

14.9.1.2 public static final String DOMFACTORYNAME = "javax.xml.parsers.DocumentBuilderFactory"

Filename containing the DOM Parser Factory Class name. Also used as the basis for the SERVICE_PID registration property.

14.9.1.3 public static final String PARSER_NAMESPACEAWARE = "parser.namespaceAware"

Service property specifying if factory is configured to support namespace aware parsers. The value is of type Boolean.

14.9.1.4 public static final String PARSER_VALIDATING = "parser.validating"

Service property specifying if factory is configured to support validating parsers. The value is of type Boolean.

14.9.1.5 public static final String SAXCLASSFILE = "/META-INF/services/javax.xml.parsers.SAXParserFactory"

Fully qualified path name of SAX Parser Factory Class Name file

14.9.1.6 public static final String SAXFACTORYNAME = "javax.xml.parsers.SAXParserFactory"

Filename containing the SAX Parser Factory Class name. Also used as the basis for the SERVICE_PID registration property.

14.9.1.7 public XMLParserActivator( )

14.9.1.8 public Object getService(Bundle bundle, ServiceRegistration registration)

bundle The bundle using the service.

registration The ServiceRegistration object for the service.

□ Creates a new XML Parser Factory object.

A unique XML Parser Factory object is returned for each call to this method.

The returned XML Parser Factory object will be configured for validating and namespace aware support as specified in the service properties of the specified ServiceRegistration object. This method can be overridden to configure additional features in the returned XML Parser Factory object.

Returns A new, configured XML Parser Factory object or null if a configuration error was encountered

14.9.1.9 public void setDOMProperties(DocumentBuilderFactory factory, Hashtable props)

factory - the DocumentBuilderFactory object

props - Hashtable of service properties.

- Set the customizable DOM Parser Service Properties.

  This method attempts to instantiate a validating parser and a namespaceaware parser to determine if the parser can support those features. The appropriate properties are then set in the specified props object.

  This method can be overridden to add additional DOM2 features and properties. If you want to be able to filter searches of the OSGi service registry, this method must put a key, value pair into the properties object for each feature or property. For example, properties.put("http://www.acme.com/features/foo", Boolean.TRUE);

14.9.1.10 public void setSAXProperties(SAXParserFactory factory, Hashtable properties)

factory - the SAXParserFactory object

properties - the properties object for the service

- Set the customizable SAX Parser Service Properties.

  This method attempts to instantiate a validating parser and a namespaceaware parser to determine if the parser can support those features. The appropriate properties are then set in the specified properties object.

  This method can be overridden to add additional SAX2 features and properties. If you want to be able to filter searches of the OSGi service registry, this method must put a key, value pair into the properties object for each feature or property. For example, properties.put("http://www.acme.com/features/foo", Boolean.TRUE);

14.9.1.11 public void start(BundleContext context) throws Exception

context The execution context of the bundle being started.

- Called when this bundle is started so the Framework can perform the bundle-specific activities necessary to start this bundle. This method can be used to register services or to allocate any resources that this bundle needs.

  This method must complete and return to its caller in a timely manner.

  This method attempts to register a SAX and DOM parser with the Framework's service registry.

Throws `Exception` – If this method throws an exception, this bundle is marked as stopped and the Framework will remove this bundle's listeners, unregister

all services registered by this bundle, and release all services used by this bundle.

See Also `Bundle.start`

14.9.1.12 public void stop(BundleContext context) throws Exception

context The execution context of the bundle being stopped.

- This method has nothing to do as all active service registrations will automatically get unregistered when the bundle stops.

Throws `Exception` – If this method throws an exception, the bundle is still marked as stopped, and the Framework will remove the bundle's listeners, unregister all services registered by the bundle, and release all services used by the bundle.

See Also `Bundle.stop`

14.9.1.13 public void ungetService(Bundle bundle, ServiceRegistration registration, Object service)

bundle The bundle releasing the service.

registration The `ServiceRegistration` object for the service.

service The XML Parser Factory object returned by a previous call to the `getService` method.

- Releases a XML Parser Factory object.

14.10 References

[15] *XML*
http://www.w3.org/XML

[16] *SAX*
http://www.saxproject.org/

[17] *DOM Java Language Binding*
http://www.w3.org/TR/REC-DOM-Level-1/java-language-binding.html

[18] *JAXP*
http://java.sun.com/xml/jaxp

[19] *JAR File specification, services directory*
http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html

[20] *Xerces 2 Java Parser*
http://xml.apache.org/xerces2-j

End Of Document