OSGi Service Platform, Mobile Specification Release 4

Draft March 9, 2005 9:14 am

OSGi Service Platform Mobile Specification

The OSGi Alliance

Release 4
August 2005

IOS Press
Ohmsha

Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

Table Of Contents

The Open Services Gateway Initiative
Bishop Ranch 2
2694 Bishop Drive
Suite 275
San Ramon
CA 94583 USA

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

LEGAL NOTICE

PRINTED IN THE NETHERLANDS
LEGAL TERMS AND CONDITIONS REGARDING SPECIFICATION

Trademarks

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".

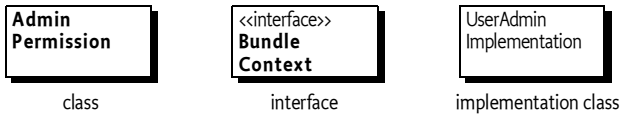
Service objects are registered with the OSGi Framework. Registration consists of the service object, a set of properties, and a list of classes and interfaces implemented by this service object. The classes and interfaces are used for type safety *and* naming. Therefore, it is said that a service object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

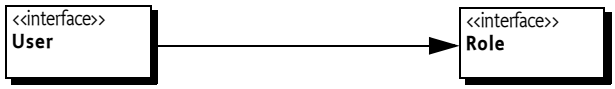


If an interface or class is used as a service object, it will have a black triangle in the bottom right corner.

Figure 2 *Service symbol*

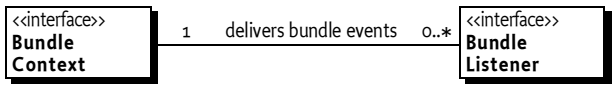
Inheritance (the extends or implements keyword in Java class definitions) is indicated with an arrow. Figure 3 shows that User implements or extends Role.

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."

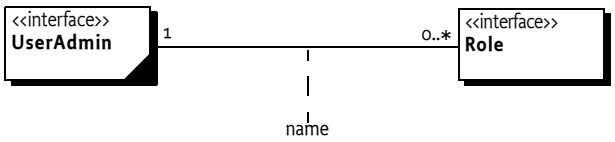
Figure 4 *Relations symbol*



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.

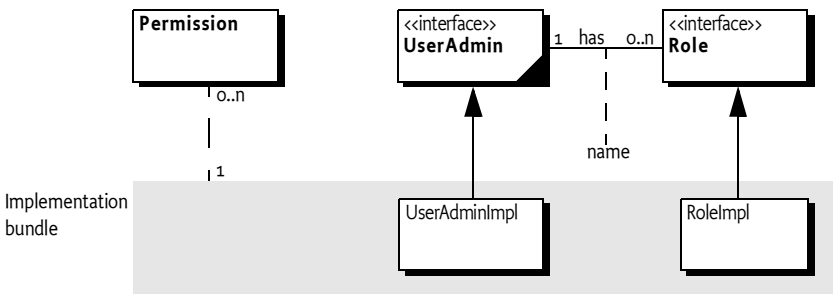
When a relationship is qualified by a name or an object, it is indicated by drawing a dotted line perpendicular to the relation and connecting this line to a class box or a description. Figure 5 shows that the relationship between a UserAdmin class and a Role class is qualified by a name. Such an association 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	1.3
Configuration Admin service	org.osgi.service.cm	1.2
Deployment Admin	org.osgi.service.deploymentadmin	1.0
Application Model	org.osgi.service.application	1.0
Meglets	org.osgi.meglets	1.0
Device Management Tree	org.osgi.service.dmt	1.0
Mobile specific	org.osgi.util.mobile	1.0
GSM Specific	org.osgi.util.gsm	1.0
IO Connector	org.osgi.service.io	1.0
Log Service	org.osgi.service.log	1.2
Metatype	org.osgi.service.metatype	1.1
Package Admin service	org.osgi.service.packageadmin	1.2
Conditional Permission Admin service	org.osgi.service.condpermission-admin	1.0
Bundle Start Levels	org.osgi.service.startlevel	1.0
URL Stream and Content	org.osgi.service.url	1.0
Service Tracker	org.osgi.util.tracker	1.3
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.

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.

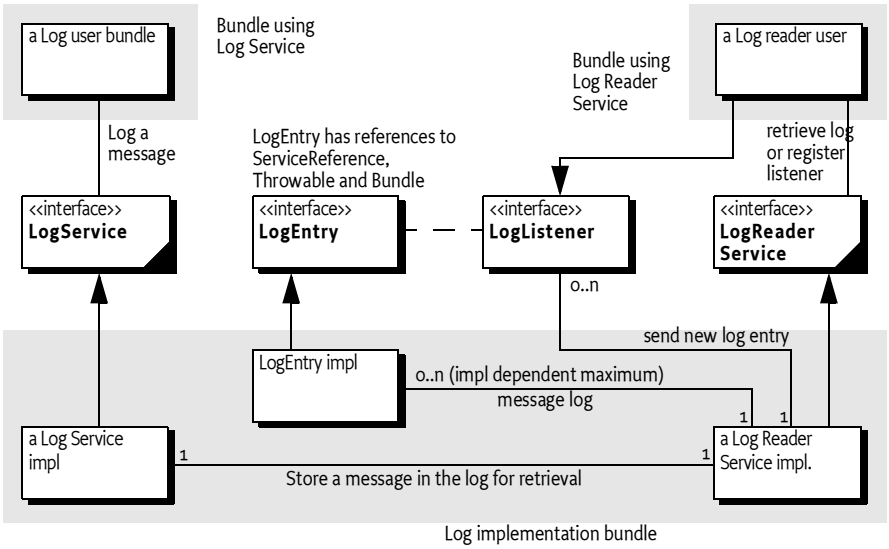
This specification defines the methods and semantics of interfaces which bundle developers can use to log entries and to retrieve log entries.

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; 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

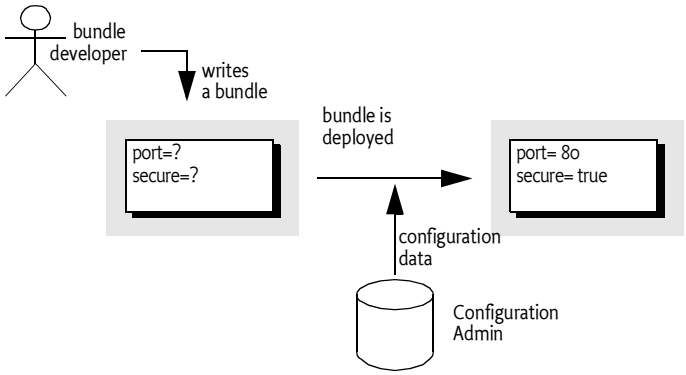
Version 1.1

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.

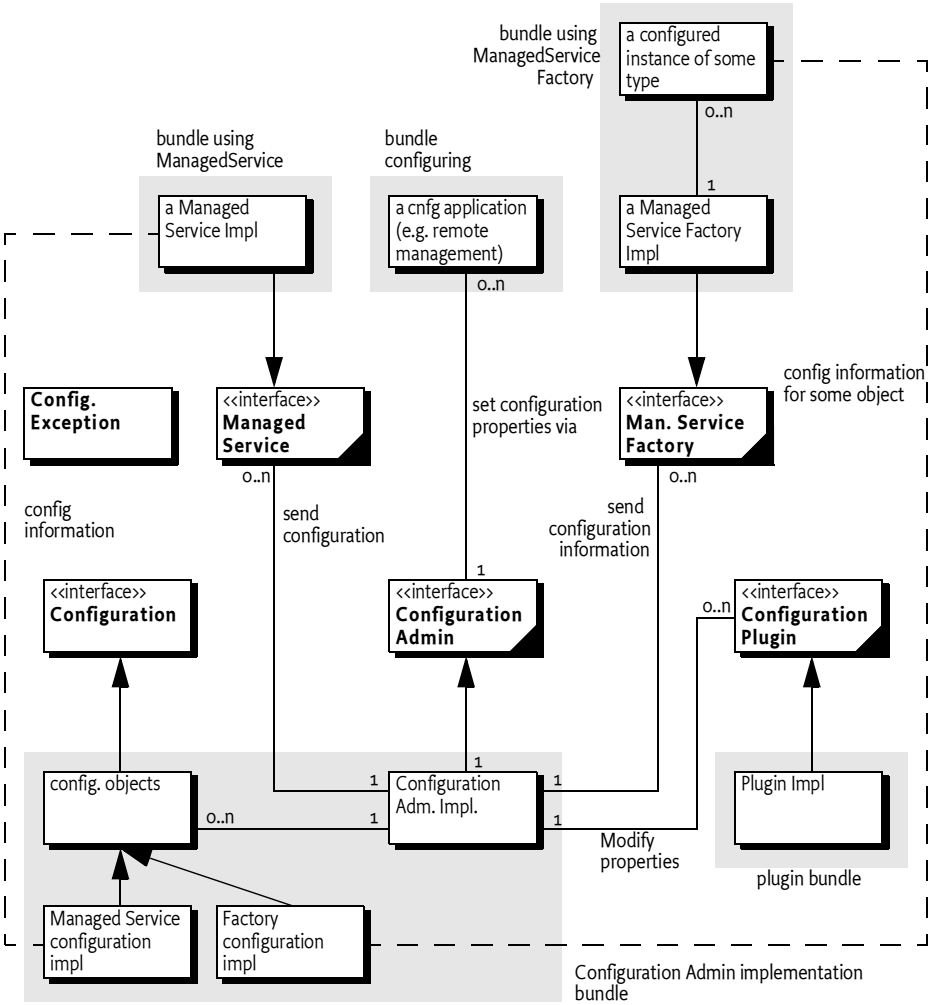
The database can be manipulated either by the Management Agent or bundles that configure themselves.

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

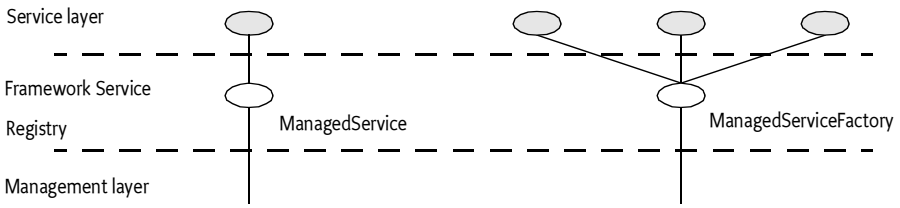
One of the more complicated aspects of this specification is the subtle distinction between the ManagedService and ManagedServiceFactory classes.

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        ::=  scalar  | primitive | vector | arrays

scalar     ::=     String     | Integer    | Long
  | Float      | Double     | Byte
  | Short      | Character  | Boolean

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

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

 vector      ::= Vector of scalar or 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 must not use nested vectors or arrays, nor must they use mixed types. Using mixed types or nesting makes it impossible to use the meta typing specification. See *Metatype Service Specification* on page 69.

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 43, 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 47.

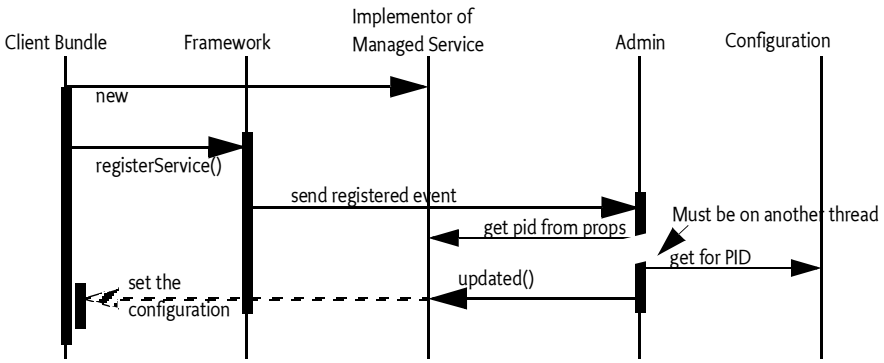
When this registration is detected by the Configuration Admin service, the following steps must occur:

- 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

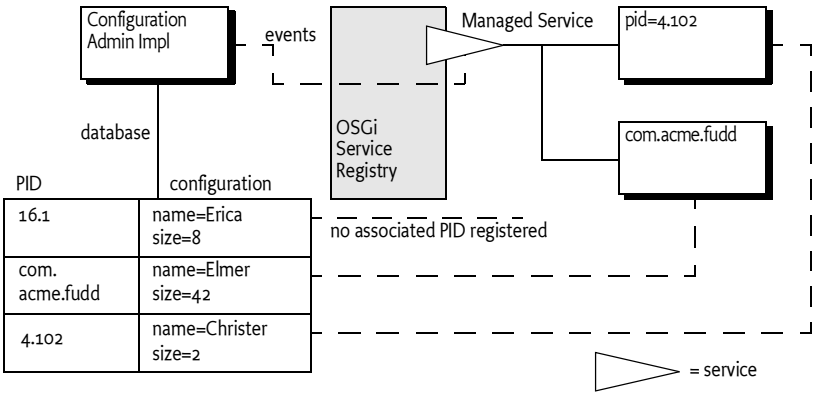
When a Managed Service is registered, the default properties may be visible in the service registry for a short period before they are replaced by the properties of the actual configuration dictionary. Care should be taken that this visibility does not cause race conditions for other bundles.

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.

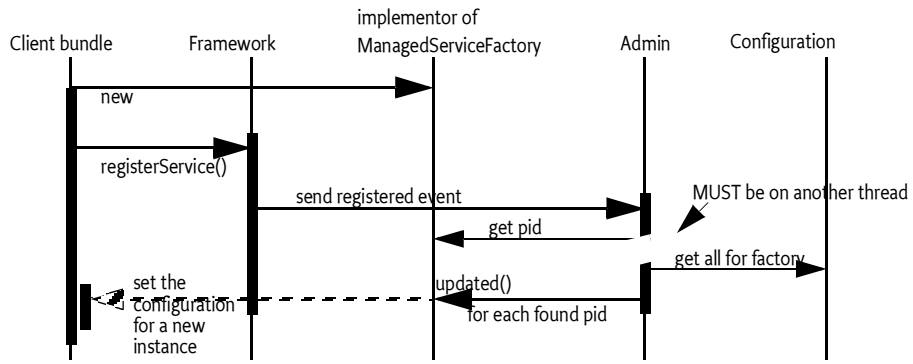
When the Configuration Admin service detects the registration of a Managed Service Factory, it must find all configuration dictionaries for this factory and must then sequentially call ManagedServiceFactory.updated(String,Dictionary) for each configuration dictionary. The first argument is the PID of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

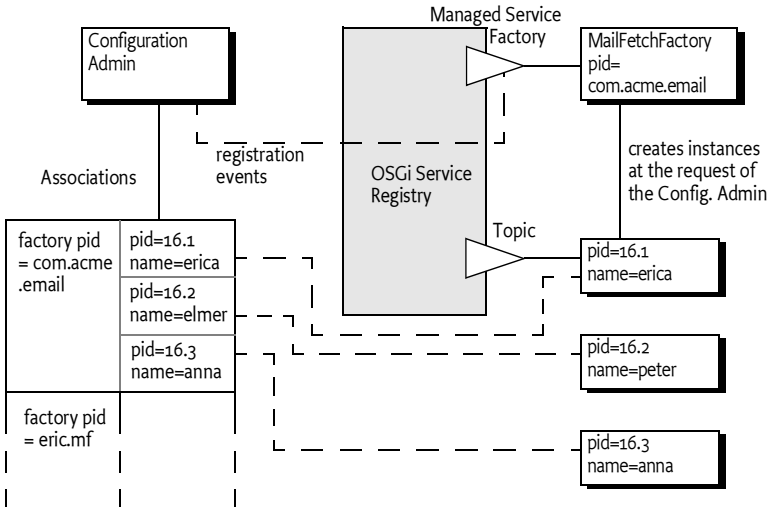
If a configuring bundle deletes an instance of a Managed Service Factory, the deleted(String) method is called. The argument is the PID for this instance. The implementation of the Managed Service Factory must remove all information and stop any behavior associated with that PID. If a service was registered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

RFC 103

3.8.1 Event Admin Service and Configuration Change Events

Configuration events are delivered asynchronously. The topic of a configuration is:

```
org/osgi/service/cm/ConfigurationEvent/<event type>
```

Event type can be any of the following:

CM_UPDATED
CM_DELETED

The properties of a configuration event are:

- cm.factoryPid – (String) The factory PID of the associated Configuration object, if the target is a Managed Service Factory. Otherwise not set.
- cm.pid – (String) The PID of the associated Configuration object.

Aren't the following service thingies not a bit overdone for CM?

- service – (ServiceReference) The Service Reference of the Configuration Admin service.
- service.id – (Long) The Configuration Admin service's ID.
- service.objectClass – (String[]) The Configuration Admin service's object class (which must include org.osgi.service.cm.ConfigurationAdmin)
- service.pid – (String) The Configuration Admin service's persistent identity

3.9 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.

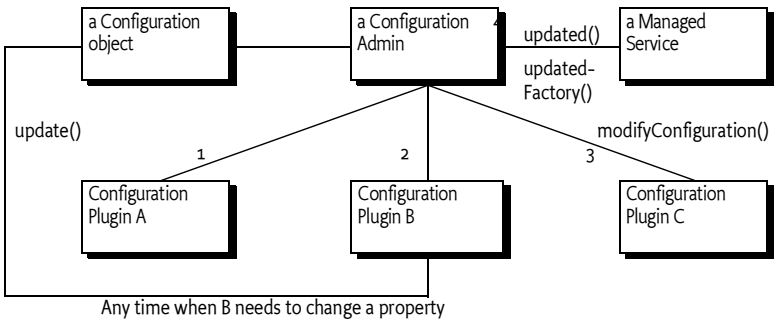
The ConfigurationPlugin interface has only one method: modifyConfiguration(ServiceReference,Dictionary). This method inspects or modifies configuration data.

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*



3.9.1 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.

3.9.2 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.

3.9.3 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.

3.9.4 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.

3.9.5 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*

3.10 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.

3.10.1 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.

3.10.2 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.

3.11 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.

3.12 Security

3.12.1 Permissions

Configuration Admin service security is implemented using Service Permission and Admin Permission. 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.

3.12.2 Configuration Permission

The Configuration Permission controls the reading and writing of configuration objects per PID.

I thought the requirement was that the name is the same as in AdminPermission?

The name parameter of the Configuration Permission specifies the target PID. The pids can be specified in the name of the permission using wildcard character at the end.

The following actions are architected:

- set – This action is needed for all the Configuration class methods and the createConfiguration method of the ConfigurationAdmin interface.
- get – This action is needed for the getConfiguration and listConfiguration methods. The listConfigurations method must only return Configuration objects to which the callers have ConfigurationPermission(pid,"get").

Shouldn't we call them modify and read?

3.12.3 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.

3.12.4 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.

3.13 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.

3.14 Changes

3.14.1 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.

3.14.2 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.

3.14.3 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 43.

3.14.4 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.

3.14.5 Equals

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

3.14.6 Constant for service.factoryPid

Added a new constant in the ConfigurationAdmin class. See *SERVICE_FACTORYPID* on page 56. This caused this specification to step from version 1.0 to version 1.1.

3.15 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; version=1.2
```

3.15.1 Summary

- Configuration - The configuration information for a ManagedService or ManagedServiceFactory object. [p.52]
- ConfigurationAdmin - Service for administering configuration data. [p.55]
- ConfigurationEvent - A Configuration Event. [p.58]
- ConfigurationException - An Exception class to inform the Configuration Admin service of problems with configuration data. [p.60]
- ConfigurationListener - Listener for Configuration Events. [p.60]
- ConfigurationPermission - Indicates a bundle's authority to set or get a Configuration. [p.61]
- ConfigurationPlugin - A service interface for processing configuration dictionary before the update. [p.62]
- ManagedService - A service that can receive configuration data from a Configuration Admin service. [p.64]
- ManagedServiceFactory - Manage multiple service instances. [p.65]

3.15.2 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.

3.15.2.1 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 an asynchronous call to all ConfigurationListeners with a ConfigurationEvent.CM_DELETED event.

Throws `IOException` – If delete fails

`IllegalStateException` – if this configuration has been deleted

3.15.2.2 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.

3.15.2.3 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.

3.15.2.4 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

3.15.2.5 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

3.15.2.6 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

3.15.2.7 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

3.15.2.8 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

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

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

3.15.2.10 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.

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.62]

3.15.3 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.65] 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.

3.15.3.1 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

3.15.3.2 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

3.15.3.3 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.54] 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.

3.15.3.4 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.54] 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.

3.15.3.5 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.

3.15.3.6 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.

3.15.3.7 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

3.15.4 public class ConfigurationEvent

A Configuration Event.

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

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

- CM_UPDATED[p.59]
- CM_DELETED[p.58]

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

See Also ConfigurationListener[p.60]

Since 1.2

3.15.4.1 public static final int CM_DELETED = 2

A Configuration has been deleted.

This ConfigurationEvent type that indicates that a Configuration object has been deleted. An event is asynchronously broadcast when a call to Configuration.delete successfully deletes a configuration.

The value of CM_DELETED is 2.

3.15.4.2 public static final int CM_UPDATED = 1

A Configuration has been updated.

This ConfigurationEvent type that indicates that a Configuration object has been updated with new properties. An event is asynchronously broadcast when a call to Configuration.update successfully changed a configuration.

The value of CM_UPDATED is 1.

3.15.4.3 public ConfigurationEvent(ServiceReference reference, int type, String factoryPid, String pid)

reference The ServiceReference object of the Configuration Admin service that created this event.

type The event type. See getType[p.59] .

factoryPid The factory pid of the associated configuration if the target of the configuration is a ManagedServiceFactory. Otherwise null if the target of the configuration is a ManagedService.

pid The pid of the associated configuration.

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

3.15.4.4 public String getFactoryPid( )

- Returns the factory pid of the associated configuration.

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

3.15.4.5 public String getPid( )

- Returns the pid of the associated configuration.

Returns Returns the pid of the associated configuration.

3.15.4.6 public ServiceReference getReference( )

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

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

3.15.4.7 public int getType( )

- Return the type of this event.

The type values are:

- CM_UPDATED[p.59]
- CM_DELETED[p.58]

Returns The type of this event.

3.15.5 public class ConfigurationException extends Exception

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

3.15.5.1 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.

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

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

reason reason for failure

cause The cause of this exception.

- Create a ConfigurationException object.

Since 1.2

3.15.5.3 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.

Since 1.2

3.15.5.4 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

3.15.5.5 public String getReason( )

- Return the reason for this exception.

Returns reason of the failure

3.15.5.6 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.

Since 1.2

3.15.6 public interface ConfigurationListener

Listener for Configuration Events.

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

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

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

Since 1.2

3.15.6.1 public void configurationEvent(ConfigurationEvent event)

event The broadcasted ConfigurationEvent object.

□ Receives notification of a broadcast ConfigurationEvent object.

3.15.7 public final class ConfigurationPermission extends BasicPermission

Indicates a bundle's authority to set or get a Configuration.

- The ConfigurationPermission.SET action allows a bundle to add, update and delete configurations for the specified names.
- The ConfigurationPermission.GET action allows a bundle to query and read configurations for the specified names.

Since 1.2

3.15.7.1 public static final String GET = "get"

The action string get (Value is "get").

3.15.7.2 public static final String SET = "set"

The action string set (Value is "set").

3.15.7.3 public ConfigurationPermission(String name, String actions)

name pid name

actions get, set (canonical order)

□ Create a new ConfigurationPermission.

The name of the service is specified as a configuration pid.

```
PID Name ::= <pid name> | <pid name ending in ".*">
```

Examples:

```
org.osgi.service.http.buffers
org.osgi.service.http.*
org.osgi.service.snmp.*
```

There are two possible actions: get and set. The set/tt› action allows a bundle to add, update and delete configurations for the specified names. The get action allows a bundle to query and read configurations for the specified names.

3.15.7.4 public boolean equals(Object obj)

obj The object to test for equality.

- Determines the equalty of two ConfigurationPermission objects. Checks that specified object has the same class name and action as this ConfigurationPermission.

Returns true if obj is a ConfigurationPermission, and has the same class name and actions as this ConfigurationPermission object; false otherwise.

3.15.7.5 public String getActions( )

- Returns the canonical string representation of the actions. Always returns present actions in the following order: get, set.

Returns The canonical string representation of the actions.

3.15.7.6 public int hashCode( )

- Returns the hash code value for this object.

Returns Hash code value for this object.

3.15.7.7 public boolean implies(Permission p)

p The target permission to check.

- Determines if a ConfigurationPermission object “implies” the specified permission.

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

3.15.7.8 public PermissionCollection newPermissionCollection( )

- Returns a new PermissionCollection object for storing ConfigurationPermission objects.

Returns A new PermissionCollection object suitable for storing ConfigurationPermission objects.

3.15.8 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.

3.15.8.1 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

3.15.8.2 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.

3.15.8.3 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.

3.15.9 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.

3.15.9.1 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

3.15.10 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 );
    }
    ...
  }
```

3.15.10.1 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.

3.15.10.2 public String getName( )

□ Return a descriptive name of this factory.

Returns the name for the factory, which might be localized

3.15.10.3 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.60] 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.

3.16 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 Service Specification

Version 1.1

Shall we remove BigDecimal and BigInteger from the java code?

4.1 Introduction

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

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.

The Meta Type Service provides a unified access point to the Meta Type information that is associated with bundles. This Meta Type information can be defined by an XML resource in a bundle, or it can be obtained from Managed Service or Managed Service Factory services that are implemented by a bundle.

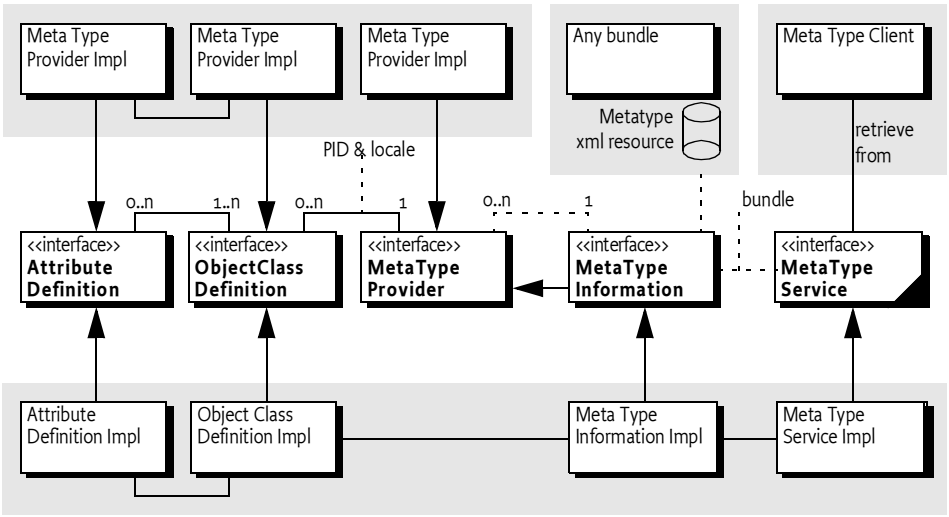
4.1.1 Essentials

- *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

- *Meta Type Service* – A service that provides a unified access point for meta type information.
- *Attribute* – A key/value pair.
- *PID* – A unique persistent ID, defined in configuration management.
- *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.
- *MetaTypeInformation* – Provides meta type information for a bundle.

Figure 16 Class Diagram Meta Typin Service, org.osgi.service.metatyping



4.1.3 Operation

The Meta Type service defines a rich dynamic typing system for properties. The purpose of the type system is to allow reasonable User Interfaces to be constructed dynamically.

The type information is normally carried by the bundles themselves. Either by implementing the MetaTypeProvider interface or by carrying one or more XML resources that define a number of Meta Types. Additionally, a Meta Type service could have other sources.

The Meta Type Service provides unified access to Meta Types that are carried by the resident bundles. The Meta Type Service collects this information from the bundles and provides uniform access to it. A client can requets the Meta Type Information associated with a particular bundle. The MetaTypeInformation object provides a list of ObjectClassDefinition objects for a bundle. 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 Classe Definitions is qualified by a locale and a Persistent IDentity (PID). This specification does not specify what the PID means. One application is OSGi Configuration Management where a PID is used by the Managed Service and Managed Service Factory services. In general, a PID should be regarded as the name of a variable where an Object Class Definition defines its type.

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:

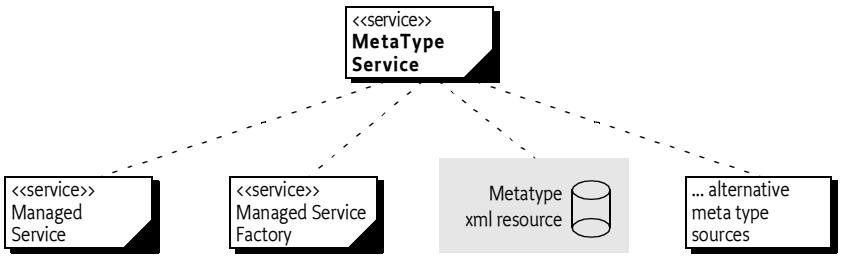
- Defined names (final ints) for the data types as restricted in the Framework for the attributes, called the syntax in OSI terms, which can be obtained with the getType() method.
- AttributeDefinition objects should use and ID that is similar to the OID as described in the ID field for ObjectClassDefinition.
- 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 Service

The Meta Type Service provides unified access to Meta Type information that is associated with a Bundle. It can get this information through the following means:

- *Meta Type Resource* – A bundle can provide one ore more XML resources that are contained in its JAR file. These resources contain and XML definition of meta types as well as to what PIDs these Meta Types apply.
- *ManagedService[Factory] objects* – As defined in the configuration management specification, ManagedService and ManagedServiceFactory service objects can optionally implement the MetaTypeProvider interface. The Meta Type Service will only search for MetaTypeProvider objects if no meta type resources are found in the bundle.

Figure 17 *Sources for Meta Types*



This model is depicted in Figure 17.

The Meta Type Service can therefore be used to retrieve meta type information for bundles which contain Meta Type resources or which provide their own MetaTypeProvider objects. The MetaTypeService interface has a single method:

- getMetaTypeInformation(Bundle) – Given a bundle, it must return the MetaTypeInformation for that bundle, even if there is no meta type information available at the moment of the call.

The returned MetaTypeInformation object maintains a map of PID to ObjectClassDefinition objects. The map is keyed by locale and PID. The list of maintained PIDs is available from the MetaTypeInformation object with the following methods:

- getPids() – PIDs for which Meta Types are available. In the Configuration Admin service
- getFactoryPids() – PIDs associated with Managed Service Factory services.

I think we discussed this but it looks like this is superfluous information, unnecessary binding the meta type to the config admin.

The MetaTypeInformation interface extends the MetaTypeProvider interface. The MetaTypeProvider interface is used to access meta type information.It supports locale dependent information so that the text used in AttributeDefinition and ObjectClassDefinition objects can be adapted to different locales.

Which locales are supported by the MetaTypeProvider object are defined by the implementor or the meta type resources.The list of available locales can be obtained from the MetaTypeProvider object.

The MetaTypeProvider interface provides the following methods:

- getObjectClassDefinition(String,String) – Get access to an ObjectClassDefinition object for the given PID. The second parameter defines the locale.
- getLocales() – List the locales.that are available.

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

```
locale = language ( '_' country ( '_' variation?) )?
```

```
language ::= < defined by ISO 3166 >
country  ::= < defined by ISO 639 >
```

For example, en, nl_BE, en_CA_posix are valid locales. The use of null for locale indicates that java.util.Locale.getDefault() must be used.

The MetaTypeService implementation class is the main class. It registers the org.osgi.service.metatype.MetaTypeService service and has a method to get a MetaTypeInformation object for a bundle.

Following is some sample code demonstrating how to print out all the ObjectClassDefinitions and AttributeDefinitions contained in a bundle:

```
void printMetaTypes( MetaTypeService mts, Bundle b ) {
   MetaTypeInformation mti =
      mts.getMetaTypeInformation(b);
   String [] pids = mti.getPids();
   String [] factoryPids = mti.getFactoryPids();
   String [] locales = mti.getLocales();

   for ( int locale = 0; locales.length; locale++ ) {
      System.out.println("Locale " + locales[locale] );
      for (int i=0; i< pids.length; i++) {
         ObjectClassDefinition ocd =
            mti.getObjectClassDefinition(pids[i], null);
         AttributeDefinitions[] ads =
            ocd.getAttributeDefinitions(ALL);
         for (int j=0; j< ads.length; j++) {
           System.out.println("OCD="+ocd.getName()
               + "AD="+ads[j].getName());
         }
      }
   }
}
```

4.6 Using the Meta Type Resources

A bundle that wants to provide meta type resources must place these resources in the META-INF/metatype, the name of the resource must be a valid JAR path. All resources in that directory must be meta type documents. Fragments can contain additional meta type resources in the same directory and they must be taken into account when the meta type resources are searched.

The meta type resources must be encoded in UTF-8.

The MetaType Service must support localization of the

- name
- icon
- description
- label attributes

The localization mechanism must be identical using the same mechanism as described in the module layer, section ####, using the same property resource. However, it is possible to override the property resource in the meta type definition resources with the localization attribute of the MetaData element.

The Meta Type Service must examine the bundle and its fragments to locate all localization resources for the localization basename. From that list, the Meta Type Service derives the list of locales which are available for the meta type information. This list can then be returned by MetaTypeInformation.getLocales method. This list can change at any time because fragments could be dynamically attached, or the bundle could be refreshed. Clients should be prepared that this list changes after they received it.

What about unresolved bundles? They do not have any fragments??

Fragments are a mess!!!

4.6.1 XML Schema of a Meta Type Resource

This section describes the schema of the meta type resource. This schema is not intended to be used during runtime for validating meta type resources. The schema is intended to be used by tools and external management systems.

The XML name space for meta type documents must be:

```
http://www.osgi.org/xmlns/metatype/v1.0.0
```

The namespace abbreviation should be metatype. I.e. the following header should be:

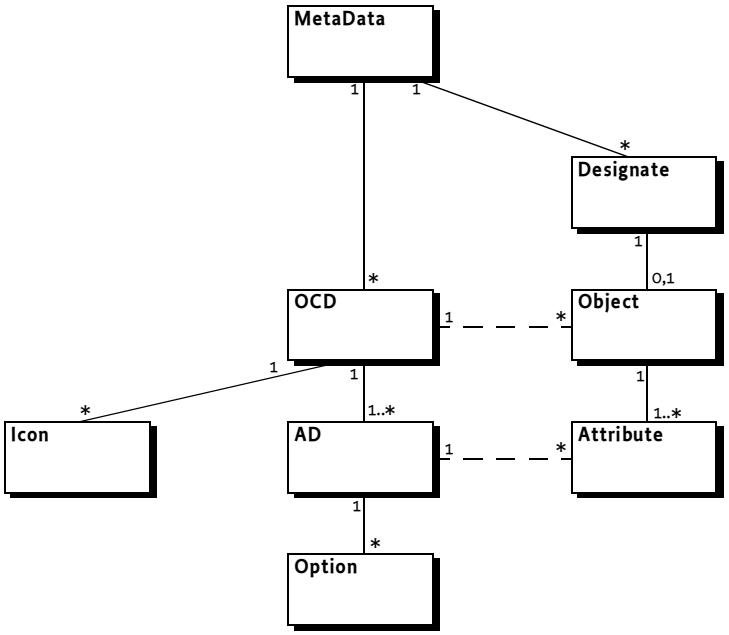
```
<metatype:MetaData
   xmlns:metatype=
      "http://www.osgi.org/xmlns/metatype/v1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
   "http://www.osgi.org/xmlns/metatype/v1.0.0 metatype.xsd
">
```

Is this schema location correct? I would expect: http://www.osgi.org/xmlns/metatype/v1.0.0/metatype.xsd

Can't we make the URLs of the schema + type equal?

The file can be found in the osgi.jar file that can be downloaded from ####.

Figure 18 *XML Schema Instance Structure (Type name = Element name)*



The element structure of the XML file is:

```
MetaData   ::= OCD* Designate*

OCD        ::= AD+ Icon*
AD         ::= Option*

Designate  ::= Object ?
Object     ::= Attribute*
```

The different elements are described in Table 9.

Attribute	Def lt	Type	Method	Description
MetaData				Top Element
localization		string		Points to the Properties file that can localize this XML. See ###
OCD				Object Class Definition
name	<>	string	getName()	A human readable name that can be localized. ### Why is the AD name optional and the OCD name required?

Table 9 *XML Schema for Meta Type resources*

	Attribute	Default	Type	Method	Description
	description			getDescription()	A human readable description of the Object Class Definition that can be localized.
	id	<>		getID()	A unique id, can not be localized.
Designate					An association between one PID and an Object Class Definition. This element *designates* a PID to be of a certain *type*.
	pid	<>	string		The PID that is associated with an OCD. This can be a factory PID or a normal PID depending on the factory attribute.
	factory		boolean		Defines if the PID should be treated as a normal variable or a factory variable.
	bundle		string		Optional Bundle Symbolic Name that implements the PID. This binds the PID to the bundle. I.e. no other bundle using the same PID may use this designation. ### Is the default the bundle the resource came from? JR> 7) Page 10-584, default for "bundle" attribute: I would say there JR> is NO default for "bundle" when it is not specified.
	optional	false	boolean		? ###
AD					Attribute Definition
	name		string	getName()	A localizable name for the Attribute Definition. description
	description		string	getDescription()	A localizable description for the Attribute Definition.
	id			getID()	The unique ID of the Attribute Definition.

Table 9 *XML Schema for Meta Type resources*

Attribute	Default	Type	Method	Description
type		string	getType()	The type of an attribute is an enumeration of the different scalar types. The string is mapped to one of the constants on the AttributeDefinition interface. Valid values, which are defined in the Scalar type, are: String ↔ STRING Long ↔ LONG Double ↔ DOUBLE Float ↔ FLOAT Integer ↔ INTEGER Byte ↔ BYTE Char ↔ CHARACTER Boolean ↔ BOOLEAN Short ↔ SHORT
cardinality	0		getCardinality()	The number of elements an instance can take. Positive numbers describe an array ([]) and negative numbers describe a Vector object.
min	### ??	string	validate(String)	A validation value. This value is not directly available from the AttributeDefinition interface. However, the validate(String) method must verify this. The semantics of this field depend on the type of this Attribute Definition. For STRING instances, ... ### what are the rules??
max	###	string	validate(String)	A validation value. Similar to the min field.
default		string	getDefaultValue()	The default value. A default is an array of String objects. The attribute must contain a comma delimited list. If the comma must be represented, it must be escaped with a back slash ('\' \u005c) . A backslash can be included with two backslashes. For example: dflt="a\,b,b\,c, c\\,d" => ["a,b", "b,c", "c\", "d"] ### yes?
Option				One option label/value for the options in an AD.

Table 9 *XML Schema for Meta Type resources*

	Attribute	Default	Type	Method	Description
	label	<>	string	getOptionLabels()	The label
	value	<>	string	getOptionValues()	The value
Icon					An icon definition.
	resource	<>	string	getIcon(int)	The resource is a URL. The base URL is assumed to be the XML file with the definition. I.e. if the XML is a resource in the JAR file, then this URL can reference another resource in that JAR file using a relative URL.
	size	<>	string	getIcon(int)	The number of pixels of the icon, maps to the size parameter of the getIcon(int) method.
Object					A definition of an instance.
	type	<>	string		A reference to the id attribute of an OCD element. I.e. this attribute defines the OCD type of this object.
Attribute					A value for an attribute of an object.
	name	<>	string		A reference to the id of the AD in the OCD as referenced by the parent Object. ### This should probably be called id ... or attributeId refId, same for Object
	content		string		The content of the attributes. If this is an array, the content must be separated by commas (',' \u002C). Commas must be escaped as described at the default attribute of the AD element. See default on page 79.

Table 9 *XML Schema for Meta Type resources*

4.6.2 Example Meta Data File

This example defines a meta type file for a Person record, based on ISO attribute types. I.e. the ids that are used are derived from ISO attributes.

```
<?xml version="1.0" encoding="UTF-8"?>
<metatype:MetaData
   xmlns:metatype=
      "http://www.osgi.org/xmlns/metatype/v1.0.0"
   xmlns:xsi=
      "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
      "http://www.osgi.org/xmlns/metatype/v1.0.0
metatype.xsd ">
```

```
<MetaData localization="person">
  <OCD name="%person" id="2.5.6.6"
     description="%Person Record">
    <AD name="%sex" id="2.5.4.12" type="Integer">
       <Option label="%male" value="1"/>
       <Option label="%Female" value="0"/>
    </AD>
    <AD name="%sn" id="2.5.4.4" type="String"/>
    <AD name="%cn" id="2.5.4.3" type="String"/>
    <AD name="%seeAlso" id="2.5.4.34" type="String"
        cardinality="8" default="http://www.google.com,
             http://www.yahoo.com"/>
    <AD name="%telNumber" id="2.5.4.20" type="String"/>
 </OCD>

 <Designate pid="com.acme.addressbook">
  <Object type="2.5.6.6"/>
 </Designate>
</MetaData>
```

Translations for this file must be stored in the same directory as this meta type resource. The property files have the root name of person. The Dutch, French and English translations could look like:

```
person_du_NL.properties:
person=Persoon
person record=Persoons beschrijving
cn=Naam
sn=Voornaam
seeAlso=Zie ook
telNumber=Tel. Nummer
sex=Geslacht
male=Mannelijk
female=Vrouwelijk

person_fr.properties
person=Person
person record=Description un person
cn=Nom
sn=Surnom
seeAlso=Reference
telNumber=Tel.
sex=Sexe
male=Homme
female=Femme

person_en_US.properties
person=Person
person record=Person Record
cn=Name
sn=Sur Name
seeAlso=See Also
telNumber=Tel.
```

```
sex=Sex
male=Male
female=Female
```

4.7 XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.osgi.org/xmlns/metatype/v1.0.0"
    xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0">
  <complexType name="MetaData">
    <sequence>
      <element name="OCD" type="metatype:OCD" minOccurs="0" maxOccurs="unbounded"/>
      <element name="Designate" type="metatype:Designate" minOccurs="1" maxOccurs="unbounded"/>
     </sequence>
   </complexType>
 <complexType name="OCD">
   <sequence>
     <element name="AD" type="metatype:AD" maxOccurs="unbounded"/>
     <element name="Icon" type="metatype:Icon" minOccurs="0"/>
    </sequence>
    <attribute name="name" type="string" use="required"/>
    <attribute name="description" type="string" use="optional"/>
    <attribute name="id" type="string" use="required"/>
  </complexType>
  <complexType name="AD">
    <sequence>
      <element name="Option" type="metatype:Option"
         minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="name" type="string" use="optional"/>
    <attribute name="description" type="string" use="optional"/>
    <attribute name="id" type="string"use="required"/>
    <attribute name="type" type="metatype:Scalar" use="required"/>
    <attribute name="cardinality" type="integer" use="optional" default="0"/>
    <attribute name="min" type="string" use="optional"/>
    <attribute name="max" type="string" use="optional"/>
    <attribute name="default" type="string" use="optional"/>
  </complexType>
  <complexType name="Object">
    <sequence>
      <element name="Attribute"type="metatype:Attribute"minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="type" type="string" use="required"/>
  </complexType>
  <complexType name="Attribute">
    <attribute name="name" type="string"/>
    <attribute name="content" type="string"/>
  </complexType>
  <complexType name="Designate">
    <sequence>
      <element name="Object" type="metatype:Object" minOccurs="0"/>
    </sequence>
    <attribute name="pid" type="string" use="required"/>
    <attribute name="factory" type="boolean" use="optional" default="false"/>
    <attribute name="bundle" type="string" use="optional"/>
    <attribute name="optional" type="boolean" default="false" use="optional"/>
  </complexType>
  <simpleType name="Scalar">
    <restriction base="string">
      <enumeration value="String"/>
      <enumeration value="Long"/>
      <enumeration value="Double"/>
      <enumeration value="Float"/>
      <enumeration value="Integer"/>
      <enumeration value="Byte"/>
      <enumeration value="Char"/>
      <enumeration value="Boolean"/>
      <enumeration value="Short"/>
```

```
      </restriction>
    </simpleType>
    <complexType name="Toption">
      <attribute name="label" type="string" use="required"/>
      <attribute name="value" type="string" use="required"/>
    </complexType>
    <complexType name="Icon">
      <attribute name="resource" type="string" use="required"/>
      <attribute name="size" type="positiveInteger" use="required"/>
    </complexType>
    <element name="MetaData" type="metatype:MetaData"/
  </schema>
```

4.8 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.

4.9 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.

4.10 Security Considerations

Special security issues are not applicable for this specification.

4.11 Changes

This specification has been changed since the previous release.

####

4.12 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; version=1.1
```

4.12.1 Summary

- AttributeDefinition - An interface to describe an attribute. [p.84]
- MetaTypeInformation - A MetaType Information object is created by the MetaTypeService to return meta type information for a specific bundle. [p.87]
- MetaTypeProvider - Provides access to metatypes. [p.87]
- MetaTypeService - The MetaType Service can be used to obtain meta type information for a bundle. [p.88]
- ObjectClassDefinition - Description for the data type information of an objectclass. [p.72]

4.12.2 public interface AttributeDefinition

An interface to describe an attribute.

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

4.12.2.1 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

4.12.2.2 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

4.12.2.3 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().

4.12.2.4 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.

4.12.2.5 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.

4.12.2.6 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.

4.12.2.7 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.

4.12.2.8 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.

4.12.2.9 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.

4.12.2.10 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.

4.12.2.11 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.

4.12.2.12 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
```

4.12.2.13 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.

4.12.2.14 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.

4.12.2.15 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

4.12.2.16 public String getName( )

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

Returns The localized name of the definition.

4.12.2.17 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

4.12.2.18 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

4.12.2.19 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.

4.12.2.20 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

4.12.3 public interface MetaTypeInformation extends MetaTypeProvider

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

4.12.3.1 public Bundle getBundle( )

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

Returns Bundle for which this object provides metatype information.

4.12.3.2 public String[] getFactoryPids( )

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

Returns Array of Factory PIDs.

4.12.3.3 public String[] getPids( )

- Return the PIDs (for ManagedServices) for which ObjectClassDefinition information is available. ### how dynamic is this? I.e. what happens when a bundle register/unregisters ### a Managed Service?

Returns Array of PIDs.

4.12.4 public interface MetaTypeProvider

Provides access to metatypes.

4.12.4.1 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.

4.12.4.2 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

4.12.5 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.

If the bundle does not contain any meta type documents, then the MetaType Service will query whether the bundle has registered any ManagedService or ManagedServiceFactory services which implement MetaTypeProvider. If so, then the MetaType Service will return a MetaTypeInformation object which wrappers these MetaTypeProvider objects. Thus the MetaType Service can be used to retrieve meta type information for bundles which contain a meta type resource or which provide their own MetaTypeProvider objects.

4.12.5.1 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. ### will this always return an object? Or can it return null. I assume it is always?

4.12.6 public interface ObjectClassDefinition

Description for the data type information of an objectclass.

4.12.6.1 public static final int ALL = -1

Argument for getAttributeDefinitions(int).

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

4.12.6.2 public static final int OPTIONAL = 2

Argument for getAttributeDefinitions(int).

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

4.12.6.3 public static final int REQUIRED = 1

Argument for getAttributeDefinitions(int).

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

4.12.6.4 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

4.12.6.5 public String getDescription( )

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

Returns The description of this object class.

4.12.6.6 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

4.12.6.7 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.

4.12.6.8 public String getName( )

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

Returns The name of this object class.

4.13 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

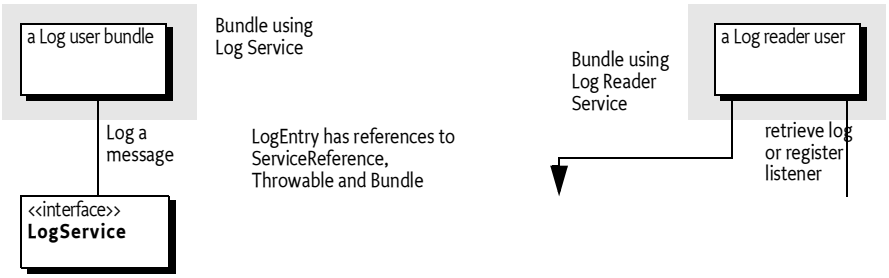
5.1 Introduction

NOT DONE YET

5.1.1 Entities

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

Figure 19 *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; version=1.0
```

5.4.1 Summary

- ComponentConstants - Defines standard names for Service Component constants. [p.92]

- ComponentContext - A ComponentContext object is used by a Service Component to interact with it execution context including locating services by reference name. [p.92]
- ComponentException - Unchecked exception which may be thrown by the Service Component Runtime. [p.94]
- ComponentFactory - When a component is declared with the factory attribute on its 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.94]
- ComponentInstance - A ComponentInstance encapsulates an instance of a component. [p.95]

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 documents 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 object is used by a Service Component to interact with it execution context including locating services by reference name.

A component’s implementation class may optional implement an activate method:

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

If a component implements this method, this method will be called when the component is activated to provide the component’s ComponentContext object.

A component’s implementation class may optional implement a deactivate method:

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

If a component implements this method, this method will be called when the component is deactivated.

The activate and deactivate methods will be called using reflection and must be at least protected accessible. These methods do not need to be public methods so that they do not appear as public methods on the component's provided service object. The methods will be located by looking through the component's implementation class hierarchy for the first declaration of the method. If the method is declared protected or public, the method will be called.

5.4.3.1 public void disableComponent(String name)

name The 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 The 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 The 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 its 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 instance.

- 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.

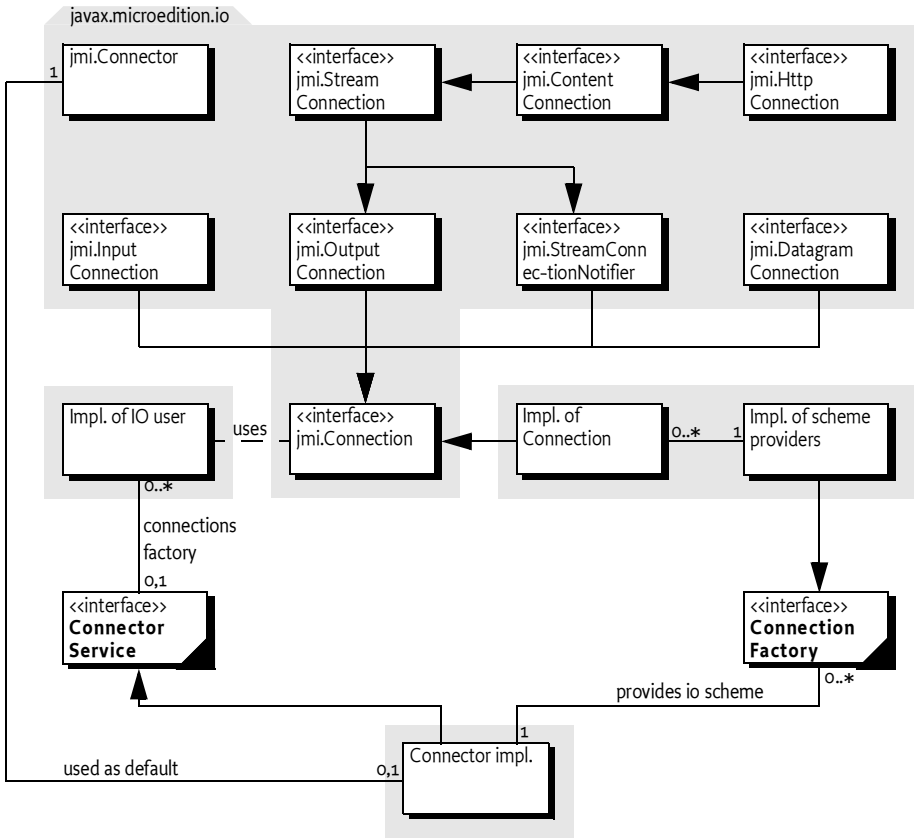
6.1.1 Essentials

- *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 20 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; 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.101]
- ConnectorService - The Connector Service should be called to create and open javax.microedition.io.Connection objects. [p.103]

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.103] 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`

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 Admin Service Specification

Version 1.0

7.1 Introduction

Nearly all the bundles in an OSGi framework must deal with events, either as an event publisher or as an event handler. So far, the preferred mechanism to disperse those events have been the service interface mechanism. I.e. dispatching events for a design related to X, usually involves a service of type XListener. However, this model does not scale well for fine grained events that must be dispatched to many different handlers. Additionally, the dynamic nature of the OSGi environment introduces several complexities because both event publishers and event handlers can appear and disappear at any time.

The Event Admin service provides an inter-bundle communication mechanism. It is based on a event *publish* and *subscribe* model, popular in many message based systems.

This specification defines the details for the participants in this event model.

7.1.1 Essentials

- *Simplifications* – The model must significantly simplify the process of programming an event source and an event handler.
- *Dependencies* – Handle the myriad of depencies between event sources and event handlers for proper cleanup.
- *Synchronicity* – It must be possible to deliver events asynchronously or synchronously with the caller.
- *Thread Usage* – It must be possible to limit the number of threads used for asynchronous event delivery. The number of event threads should be configurable.

Is the nr of threads in the API? Otherwise remove this essential

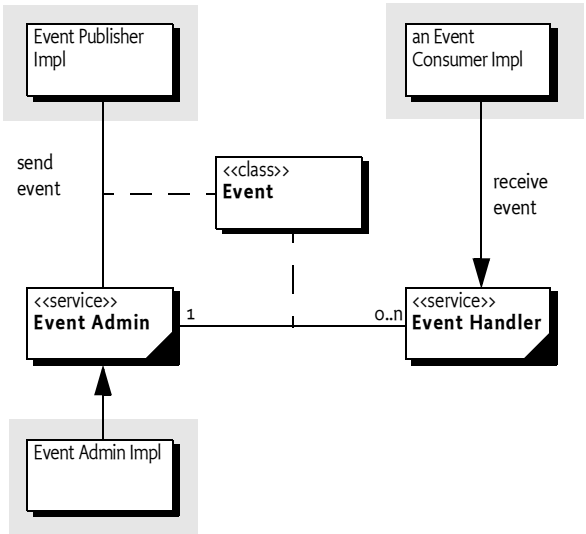
- *Event Window* – Only event handlers that are active when an event is published must receive this event, handlers that register later must not see the event.
- *Performance* – The event mechanism must impose minimal overhead in delivering events.
- *Selectivity* – Event listeners must only receive notifications for the event types for which they are interested
- *Reliability* – The Event Admin must ensure that events continue to be delivered regardles the quality of the event handlers.

- *Security* – Publishing and receiving events are sensitive operations that must be protected per event type.
- *Extendability* – It must be possible to define new event types with their own data types.
- *Native Code* – Events must be able to be passed to native code or come from native code.
- *OSGi Events* – The OSGi Framework,as well as a number of OSGi services, already have number of its own events defined. For uniformity of processing, these have to be mapped into generic event types.

7.1.2 Entities

- *Event* – An Event object has a topic and a Dictionary object that contains the event properties. It is an immutable object.
- *Event Admin* – The service that provides the publish and subscribe model to Event Handlers and Event Publishers.
- *Event Handler* – A service that receives and handles Event objects.
- *Event Publisher* – A bundle that sends event through the Event Admin service.
- *Event Subscriber* – Another name for an Event Handler.
- *Topic* – The name of an Event type.
- *Event Properties* – The set of properties that is associated with an Event.

Figure 21 *The Event Admin service org.osgi.service.event package*



7.1.3 Synopsis

The Event Admin service provides a point for bundles to publish events, regardless of their destination. It therefore also accessed by Event Handlers to subscribe to specific types of events.

Events are published under a topic, together with a number of event properties. Event Handlers can specify a filter to control the Events they receive on a very fine grained basis.

7.1.4 What To Read

- *Architects* – The *Event Handling Patterns* on page 109 section provides an overview of the patterns used in the Event Admin service.
- *Event Publishers* – The *Event Publisher* on page 113 provides an introduction how to write an Event Publisher. The *Event Handling Patterns* on page 109 provides a good overview of the patterns used.
- *Event Subscribers/Handlers* – The *Event Handler* on page 112 provides the rules how to subscribe and handle events.

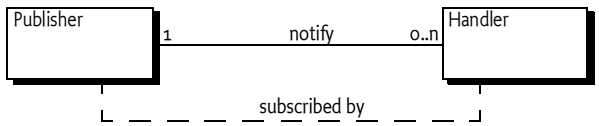
7.2 Event Handling Patterns

The Event Admin service provides very common functionality that can be found in many different systems. This section discusses different patterns that are applicable in this area.

7.2.1 Observer Pattern

The Observer pattern defines a one-to-many dependency between a publisher and the handler that must be notified when the source's state changes. Each handler registers itself with the publisher. When the publisher's state changes, it notifies each registered handler. This couples the publisher and the handler because each must know about the other. This pattern is shown in Figure 22.

Figure 22 *Observer Pattern*

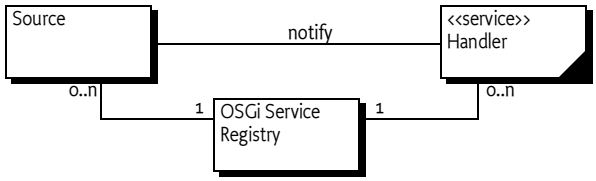


In an OSGi Service Platform, either the publisher or the handler may be in a bundle that could be removed at any time. Since the Framework is unaware of the handler's dependency it cannot clean it up automatically. Therefore both the publisher and the handler must monitor the other and take appropriate action if the other is removed from the system. In practice, getting this right is complicated.

7.2.2 Whiteboard Pattern

The *Whiteboard* pattern makes use of the OSGi service registry to manage the dependencies between the publisher and its handlers. Each handler registers itself as a service in the OSGi service registry. The publisher queries the registry to obtain the list of handlers, then dispatches the event directly to each one of them. This pattern is depicted in Figure 23.

Figure 23 *Whiteboard Pattern*

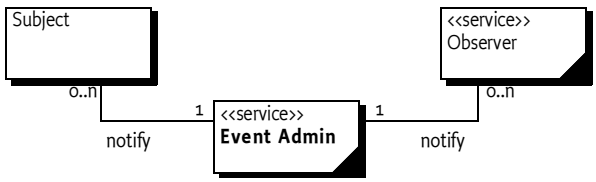


In this model, the publisher and the handler are less coupled than before because the handler has no direct knowledge of the publisher. However, the publisher must still monitor the service registry and take action when a handler is either registered or unregistered. The OSGi implementations provide the ServiceTracker utility class to handle the necessary details.

7.2.3 Publish Subscribe

The *Publish-Subscribe* pattern decouples sources from their handlers by interposing an *event channel* between them. The publisher posts events to the channel, which identifies which handlers need to be notified and then takes care of the notification process. This model is depicted in Figure 24.

Figure 24 *Channel Pattern*



In this model, the event source and event handler are completely decoupled because neither has any direct knowledge of the other. The complicated logic of monitoring changes in the event publishers and event handlers is completely contained within the event channel. This is highly advantageous in an OSGi environment because it simplifies the process of both sending and receiving events.

There are two main flavors of publish-subscribe mechanisms: *subject-based* and *content-based*. In subject-based systems, handlers register to receive events that were sent to a particular topic (or set of topics). Content-based systems examine the entire contents of the event to determine which handlers should receive it. They are more flexible than subject-based systems but the rules for matching events to handlers are more complicated so some thought needs to be given to the efficiency of the system.

7.2.4 Asynchronous Event Delivery

Asynchronous event delivery creates additional complications because event publishers and event handlers may disappear from the Framework between the time that the event is triggered and delivery is actually attempted.

Also, asynchronous event delivery implies the use of one or more event threads. Bundles that need to asynchronously deliver events generally provide their own event thread(s). In a worst-case scenario, every bundle that is installed in the framework may need to create its own. This can lead to a significant overhead to manage.

A general event dispatch mechanism offers a chance to consolidate all of those threads into just a single one, or at least a manageable number of threads, saving a fair amount of resources.

7.2.5 Security

The process of delivering events necessarily involves passing messages (and therefore data) between different bundles. Handlers are generally invoked within the context of the event publisher, and are therefore limited by its (lack of) privileges. If a handler must perform some sensitive operationit is recommeded to use the doPrivileged method to ensure that the handler executes with its full privileges not being restricted by the event source. Knowing when and how to use doPrivileged cannot be considered simple. The Event Channel model can take care of this security handling.

7.3 The Event

Events have the following attributes:

- *Topic* – A topic that defines what happened. For example, when a bundle is started an event is published that has a topic of org/osgi/framework/BundleEvent/STARTED.
- *Properties* – Zero or more properties that contain additional information about the event. For example, the previous example event has a property of bundle.id which is set to a Long object, among other properties.

7.3.1 Topics

The topic of an event defines the *type* of the event. It is fairly granular in order to give handlers the opportunity to register for just the events they are interested in. When a topic is designed, its name should not include any other information, such as the publisher of the event or the data associated with the event, those parts are intended to be stored in the event properties.

The topic is intended to serve as a first-level filter for determining which handlers should receive the event. Event Admin service implementations use the structure of the topic to optimize the dispatching of the events to the handlers.

Topics are arranged in a hierarchical namespace. Each level is defined by a token and levels are separated by slashes. More precisely, the topic must conform to the following grammar:

```
topic ::= token ( '/' token ) *
```

Topics should be designed to become more specific when going from left to right. Handlers can provide a prefix that matches a topic, using the preferred order allows a handler to minimize the number of prefixes it needs to register.

Topics are case sensitive. As a convention, topics should follow the reverse domain name scheme used by Java packages to guarantee uniqueness.

But no dots then? Is this a valid recommendation?

This specification uses the convention fully/qualified/package/ClassName/ACTION. If necessary a psuedo-class-name is used.

7.3.2 Properties

Information about the actual event is provided as properties. The property name is a case-sensitive string and the value can be any object. Although any Java object can be used as a property value, only String objects and the eight primitive types (plus their wrappers) should be used. Other types cannot be passed to handlers that reside external from the Java VM.

Another reason that arbitrary classes should not be used is the mutability of objects. If the values are not immutable, then any handler that receives the event could change the value. Any handlers that received the event subsequently would see the altered value and not the value as it was when the event was sent.

The topic of the event is available as a property with the key EVENT_TOPIC. This allows filters to include the topic as a condition if necessary.

7.4 Event Handler

Event handlers must be registered as services with the OSGi framework under the object class org.osgi.service.event.EventHandler.

Event handlers should be registered with a property (constant from EventConstants) EVENT_TOPIC. The value being a String[] object that describes which *topics* the handler is interested in. A wildcard ('*' \u002A) may be used at the end of the topic name to match only the prefix of a topic. Event Handlers which do not have a value for the topic property are treated as though they had the value {"*"}, i.e. they match any topic. Such a promiscuous filter should be used with care because it reduces the possibilities for the Event Admin service to optimize dispatching events.

I think the topic should be mandatory, if you do not set it, you do not get anything?

More precisely, the value of each entry in the EVENT_TOPIC service registration property must conform to the following grammar:

```
topic-scope ::= '*' | ( topic [ '/*' ] )
```

Event handlers can also be registered with a service property named EVENT_FILTER. The value of this property must be a string containing a Framework filter specification. Any of the event's properties can be used in the filter expression.

Each Event Handler is notified for any event which belongs to the topics the handler has expressed an interest in. If the handler has defined a filter service property then the event properties must also match the filter expression.

For example, a bundle wants to see all Log Service events with a level of WARNING or ERROR, but only, i.e. it must ignore the INFO and DEBUG events. Additionally, the only events of interest are when the bundle symbolic name starts with com.acme.

```
public AcmeWatchDog implements Activator, EventHandler {
   final static String [] topics = new String[] {
      "org.osgi.service.log.LogService.LOG_WARNING",
      "org.osgi.service.log.LogService.LOG_ERROR" };

   public void start(BundleContext context) {
      Dictionary d = new Hashtable();
      d.put(EventConstants.EVENT_TOPICS, topics );
      d.put(EventConstants.EVENT_FILTER,
         "(bundle.symbolicName=com.acme.*)" );
      context.registerService( EventHandler.class.getName(),
         this, d );
   }
   public void stop( BundleContext context) {}

   public void handleEvent(Event event ) {
      //...
   }
}
```

Check!

7.5 Event Publisher

To fire an event, the event source must retrieve the Event Admin service from the OSGi service registry. Then it creates the event object and calls one of the event admin's methods to fire the event either synchronously or asynchronously.

The following example is a class that publishes a time event every 60 seconds.

```
public class TimerEvent extends Thread
   implements BundleActivator {

   ServiceTrackertracker;

   public TimerEvent() { super("TimerEvent"); }

   public void start(BundleContext context ) {
      tracker = new ServiceTracker(context,
         EventAdmin.class.getName(), null );
      start();
   }

   public void stop( BundleContext context ) {
      interrupt();
   }
```

```
   public void run() {
      while ( ! Thread.interrupted() ) try {
         Calendarc = Calendar.getInstance();
         Dictionaryp = new Hashtable();
         p.put("minutes",new
Integer(c.get(Calendar.MINUTE)));
            p.put("hours",
               new Integer(c.get(Calendar.HOUR_OF_DAY)));
            p.put("dayMonth", new
Integer(c.get(Calendar.DAY_OF_MONTH)));
            p.put("dayWeek", new
Integer(c.get(Calendar.DAY_OF_WEEK)));
            p.put("dayYear", new
Integer(c.get(Calendar.DAY_OF_YEAR)));
         EventAdminea = (EventAdmin) tracker.getService();
         if ( ea != null ) {
            ea.sendEvent(new Event("com.acme.timer", p ));
         }
         Thread.sleep(60000-c.get(Calendar.SECOND)*1000);
      } catch( InterruptedException e ) {
         // ignore, treated by while loop
      }
   }

   public static void main(String args[] ) {
      TimerEvent te = new TimerEvent();
      te.start();

   }
}
```

7.6 Specific Events

7.6.1 General Conventions

Some handlers are more interested in the contents of an event rather than what actually happened. For example, a handler needs to be notified whenever an Exception is thrown anywhere in the system. Both Framework Events and Log Entry events may contain an exception that would be of interest to this hypothetical handler. If both FrameworkEvents and LogEntrys use the same property names then the listener can access the Exception in exactly the same way. If some future event type follows the same conventions then the handler can receive and process the new event type even though it had no knowledge of it when it was compiled.

The following properties are suggested as conventions. When new event types are defined they should use these names with the corresponding types and values where appropriate. These values should be set only if they are not null

A list of these property names can be found in Table 10..

Name	Type	Notes
bundle.symbolicName	String	A bundle's symbolic name
event	Object	The actual event object. Used when rebroadcasting an event that was sent via some other event mechanism
exception	Throwable	An exception or error
exception.message	String	Must be equal to exception.getMessage()
message	String	A human-readable message ### Which locale
service.objectClass	String[]	A service's objectClass ### We already have a constant for objectclass?
service	ServiceReference	A service
service.id	Long	A service's id
service.pid	String	A service's persistent identity
timestamp	Long	The time when the event occurred, as reported by System.currentTimeMillis()
bundle.signer	String	A signer DN ### Yes?

Table 10 *General property names for events*

Are we going to provide constants for these

The topic of an OSGi event is constructed by taking the fully qualified name of the event class, substituting a slash for every period, and appending a slash followed by the name of the constant that defines the event type. For example, the topic of

```
BundleEvent.STARTED
```

Event becomes

```
org/osgi/framework/BundleEvent/STARTED
```

If a type code for the event is unknown then the event must be ignored.

Should we not make the advise to use the class name for event topic general?

7.6.2 OSGi Events

In order to present a consistent view of all the events occurring in the system, the existing Framework-level events are mapped to the Evant Admin's publish-subscribe model. This allows event subscribers to treat framework events exactly the same as other events.

The properties associated with the event depends on its class as outlined in the following sections.

7.6.3 Framework Event

Framework Events must be delivered asynchronously with a topic of:

```
org/osgi/framework/FrameworkEvent/<event type>
```

The following event types are supported:

```
STARTED                 1
ERROR                   2
PACKAGES_REFRESHED      4
STARTLEVEL_CHANGED      8
```

Other events are ignored, no event will be send by the Event Admin. The following event properties must be set for a Framework Event.

- event – (FrameworkEvent) The original event object.

If the FrameworkEvent getBundle method returns a non null value, the following fields must be set:

- bundle.id – (Long) The source's bundle id.
- bundle.symbolicName – (String) The source bundle's symbolic name. Only set if the bundle's symbolic name is not null.
- bundle – (Bundle) The source bundle.

If the FrameworkEvent getThrowable method returns a non null value:

- exception.class – (String) The fully-qualified class name of the attached Exception.
- exception.message –(String) The message of the attached exception. Only set if the Exception message is not null.
- exception – (Throwable) The Exception returned by the getThrowable method.

7.6.4 Bundle Event

Framework Events must be delivered asynchronously with a topic of:

```
org/osgi/framework/BundleEvent/<event type>
```

The following event types are supported:

```
INSTALLED               1
STARTED                 2
STOPPED                 4
UPDATED                 8
UNINSTALLED             16
```

Other events are ignored, no event will be send by the Event Admin. The following event properties must be set for a Bundle Event. If listeners require synchronous delivery then they should register a Synchronous Bundle Listener with the Framework.

- event – (BundleEvent) The original event object.
- bundle.id – (Long) The source's bundle id.
- bundle.symbolicName – (String) The source bundle's symbolic name. Only set if the bundle's symbolic name is not null.

- bundle – (Bundle) The source bundle.

7.6.5 Service Event

Service Events must be delivered synchronously with the topic:

org/osgi/framework/ServiceEvent/<event type>

The following event types are supported:

REGISTERED	1
MODIFIED	2
UNREGISTERING	3

Sure this is a wise idea? I do not like the post method ... I suggest we keep the synchronous methods on the framework

- event – (BundleEvent) The original event object.
- service – (ServiceReference) The result of the getServiceReference method
- service.id – (Long) The service's ID.
- service.pid – (String) The service's persistent identity. Only set if not null.
- service.objectClass – (String[]) The service's object class.

7.6.6 Log Events

Log events must be delivered asynchronously under the topic:

org/osgi/service/log/LogEntry/<event type>

Dont we need the log levels?

The logging level is used as event type:

LOG_ERROR	1
LOG_WARNING	2
LOG_INFO	3
LOG_DEBUG	4

The properties of a log event are:

- bundle.id – (Long) The source bundle's id.
- bundle.symbolicName – (String) The source bundle's symbolic name. Only set if not null.
- bundle – (Bundle) The source bundle.
- log.level – (Integer) The log level.
- message – (String) The log message.
- timestamp – (Long) The log entry's timestamp.
- log.entry – (LogEntry) The LogEntry object.

If the log entry has an associated Exception:

- exception.class – (String) The fully-qualified class name of the attached exception. Only set if the getExceptionmethod returns a non-null value.
- exception.message – (String) The message of the attached Exception. Only set if the Exception message is not null.
- exception – (Throwable) The Exception returned by the getException method.

If the getServiceReference method returns a non-null value:

- service – (ServiceReference) The result of the getServiceReference method.
- service.id – (Long) The id of the service.
- service.pid – (String) The service's persistent identity. Only set if the service.pid service property is not null.
- service.objectClass – (String[]) The object class of the service object.

7.7 Event Admin Service

The Event Admin service must be registered as a service with the object class org.osgi.service.event.EventAdmin. The Event Admin service should be a singleton.

This is currently under discussion

The Event Admin service is responsible for tracking the registered handlers, handling event notifications and providing a thread for asynchronous event delivery.

7.7.1 Synchronous Event Delivery

Synchronous event delivery is initiated by the sendEvent method. When this method is invoked, the Event Admin service determines which handlers must be notified of the event and then notifies each one in turn. The handlers can be notified in the caller's thread or in an event-delivery thread, depending on the implementation. In either case, all notifications must be completely handled before the sendEvent method returns to the caller.

Synchronous event delivery is significantly more expensive than asynchronous delivery. All things considered equal, the asynchronous delivery should be preferred over the synchronous delivery.

Callers of this method will need to be coded defensively and assume that synchronous event notifications could be handled in a separate thread. That entails that they must not be holding any monitors when they invoke the sendEvent method. Otherwise they significantly increase the likelyhood of deadlocks because Java monitors are not reentrant from another thread by definition. Not holding monitors is good practice even when the event is dispatched in the same thread.

7.7.2 Asynchronous Event Delivery

Asynchronous event delivery is initiated by the postEvent method. When this method is invoked, the Event Admin service must determine which handlers are interested in the event. By collecting this list of listeners during the method invocation, the Event Admin service ensures that only handlers that were registered at the time the event was posted will receive the event notification.

Example

The Event Admin service can use more than one thread to deliver events. If it does then it must guarantee that each handler receives the events in the same order as the events were posted. This ensures that handlers see events in the expected order. For example, it would be an error to see a `destroyed` event before the corresponding `created` event.

Before notifying each handler, the event delivery thread must ensure that the handler is still registered in the service registry. If it has been unregistered then the handler must not be notified.

The Event Admin service ensures that events are delivered in a well-defined order. For example, if a thread posts events `A` and `B` in the same thread then the handlers should not receive them in the order `B`, `A`. if A and B are posted by different threads at about the same time then no guarantees about the order of delivery are made.

Sequence diagram

7.7.3 Order of Event Delivery

Asynchronous events are delivered in the order in which they arrive in the event queue. Thus if two events are posted by the same thread then they will be delivered in the same order (though other events may come between them). However, if two or more events are posted by different threads then the order in which they arrive in the queue (and therefore the order in which they are delivered) will depend very much on subtle timing issues. The event delivery system cannot make any guarantees in this case.

Synchronous events are delivered as soon as they are sent. If two events are sent by the same thread, one after the other, then they must be guaranteed to be processed serially and in the same order. However, if two events are sent by different threads then no guarantees can be made. The events MAY be processed in parallel or serially, depending on whether or not the Event Admin service dispatches synchronous events in the caller's thread or in a separate thread.

?? I think we should guarantee that there is only one postEvent active at any moment in time?

Note that if the actions of a handler trigger a synchronous event, then the delivery of the first event will be paused and delivery of the second event will begin. Once delivery of the second event has completed, delivery of the first event will resume. Thus some listeners may observe the second event before they observe the first one.

Sequence diagram

7.8 Reliability

7.8.1 Exceptions in callbacks

If a handler throws an Exception during delivery of an event, it must be caught by the Event Admin service and handled in some implementation specific way. If a Log Service is available the exception should be logged. Once the exception has been caught and dealt with, the event delivery must continue with the next handlers to be notified, if any.

7.8.2 Dealing with Stalled Listeners

Event handlers should not spend too long in the handleEvent method. Doing so will prevent other handlers in the system from being notified. If a handler needs to do something that can take a while, it should do it in a different thread.

An event admin implementation can attempt to detect stalled or deadlocked handlers and deal with them appropriately. Exactly how it deals with this situation is left as implementation specific. One allowed implementation is to mark the current event delivery thread as invalid and spawn a new event delivery thread. Event delivery should resume with the next handler to be notified.

Shouldnt this be a must?

Implementations can choose to blacklist any handlers that they determine are misbehaving. Blacklisted handlers must not be notified of any events. If a handler is blacklisted, the event admin should log a message that explains the reason for it.

This needs to be cleaned up ... or we mandate it, or we list it and tell them to figure it out

7.9 Interoperability with Native Applications

Implementations of the Event Admin service can support passing events to, and/or receiving events from native applications.

If the implementation supports native interoperability, it must be able to pass the topic of the event and its properties to/from native code. Implementations must be able to support property values of the following types:

- String objects, including full Unicode support
- Integer, Long, Byte, Short, Float, Double, Boolean, Character objects
- Single-dimension arrays of the above types (including String)
- Single-dimension arrays of Java's eight primitive types (int, long, byte, short, float, double, boolean, char)

Implementations can support additional types. Property values of unsupported types must be silently discarded.

7.10 Security

7.10.1 TopicPermission

The TopicPermission class allows fine-grained control over which bundles may post events to a given topic and which bundles may receive those events.

The target parameter for the permission is the topic name. TopicPermission classes uses a wildcard matching algorithm similar to the BasicPermission class, except that slashes are used as separators instead of periods. For example, a name of a/b/* implies a/b/c but not x/y/z or a/b.

There are two available actions: PUBLISH and SUBSCRIBE. These control a bundle's ability to either publish or receive events, respectively. Neither one implies the other.

How about POST (synchronous)? Seems you would like to have a special permission for this.

7.10.2 Required Permissions

Bundles that need to register an event listener must be granted ServicePermission[org.osgi.service.event.EventHandler, REGISTER]. In addition, handlers require TopicPermission[<topic>, SUBSCRIBE] for each topic they want to be notified about.

Bundles that need to publish an event must be granted ServicePermission[org.osgi.service.event.EventAdmin, GET] so that they may retrieve the Event Admin service and use it. In addition, event sources require TopicPermission[<topic>, PUBLISH] for each topic they want to send events to.

Bundles that need to iterate the handlers registered with the system must be granted ServicePermission[org.osgi.service.event.EventHandler, GET] to retrieve the event handlers from the service registry.

Only the bundle that contains the Event Admin service implementation should be granted ServicePermission[org.osgi.service.event.EventAdmin, REGISTER] to register the event channel admin service.

7.10.3 Security Context During Event Callbacks

During an event notification, the Event Admin service's Protection Domain will be on the stack above the handler's Protection Domain. In the case of a synchronous event, the event publisher's protection domain can also be on the stack.

picture of the stack

Therefore, if a handler needs to perform a secure operation using its own privileges, it must invoke the doPrivileged method to isolate its security context from that of its caller.

The event delivery mechanism must not wrap event notifications in a doPrivileged call.

7.11 org.osgi.service.event

The OSGi Event 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.event; version=1.0
```

7.11.1 Summary

- Event - An event. [p.122]
- EventAdmin - The Event Admin service. [p.123]
- EventConstants - Defines standard names for EventHandler properties. [p.123]
- EventHandler - Listener for Events. [p.124]
- TopicPermission - A bundle's authority to publish or subscribe to event on a topic. [p.125]

7.11.2 public class Event

An event.

Event objects are delivered to EventHandler services which subsrcibe to the topic of the event.

NOTE: Although it is permitted to subclass Event, the operations defined by this class MUST NOT be overridden.

7.11.2.1 public Event(String topic, Dictionary properties)

topic The topic of the event.

properties The event's properties (may be null).

□ Constructs an event.

Throws `IllegalArgumentException` – If topic is not a valid topic name.

7.11.2.2 public boolean equals(Object object)

object The Event object to be compared.

□ Compares this Event object to another object.

An event is considered to be **equal to** another event if the topic is equal and the properties are equal.

Returns true if object is a Event and is equal to this object; false otherwise.

7.11.2.3 public final Object getProperty(String name)

name the name of the property to retrieve

□ Retrieves a property.

Returns The value of the property, or null if not found.

7.11.2.4 public final String[] getPropertyNames( )

□ Returns a list of this event's property names.

Returns A non-empty array with one element per property.

7.11.2.5 public final String getTopic( )

- Returns the topic of this event.

Returns The topic of this event.

7.11.2.6 public int hashCode( )

- Returns a hash code value for the object.

Returns An integer which is a hash code value for this object.

7.11.2.7 public final boolean matches(Filter filter)

filter The filter to test.

- Tests this event's properties against the given filter.

Returns true If this event's properties match the filter, false otherwise.

7.11.2.8 public String toString( )

- Returns the string representation of this event.

Returns The string representation of this event.

7.11.3 public interface EventAdmin

The Event Admin service. Bundles wishing to publish events must obtain the Event Admin service and call one of the event delivery methods.

7.11.3.1 public void postEvent(Event event)

event The event to send to all listeners which subscribe to the topic of the event.

- Initiate asynchronous delivery of an event. This method returns to the caller before delivery of the event is completed.

Throws `SecurityException` – If the caller does not have TopicPermission[topic, PUBLISH] for the topic specified in the event.

7.11.3.2 public void sendEvent(Event event)

event The event to send to all listeners which subscribe to the topic of the event.

- Initiate synchronous delivery of an event. This method does not return to the caller until delivery of the event is completed.

Throws `SecurityException` – If the caller does not have TopicPermission[topic, PUBLISH] for the topic specified in the event.

7.11.4 public interface EventConstants

Defines standard names for EventHandler properties.

7.11.4.1 public static final String EVENT_FILTER = "event.filter"

Service Registration property (named event.filter) specifying a filter to further select Event s of interest to a Event Handler service.

Event handlers MAY be registered with this property. The value of this property is a string containing an LDAP-style filter specification. Any of the event's properties may be used in the filter expression. Each event handler is notified for any event which belongs to the topics in which the handler has expressed an interest. If the event handler is also registered with this service property, then the properties of the event must also match the filter for the event to be delivered to the event handler.

See Also Event[p.122], org.osgi.framework.Filter

7.11.4.2 public static final String EVENT_TOPIC = "event.topics"

Service registration property (named event.topic) specifying the Event topics of interest to a Event Handler service.

Event handlers SHOULD be registered with this property. The value of the property is an array of strings that describe the topics in which the handler is interested. An asterisk ("*") may be used as a trailing wildcard. Event handlers which do not have a value for this propery are treated as though they had specified this property with the value { "*" }. More precisely, the value of each entry in the array must conform to the following grammar:

```
topic-description := "*" | topic ( "/*" )?
topic := token ( "/" token )*
```

See Also Event[p.122]

7.11.5 public interface EventHandler

Listener for Events.

EventHandler objects are registered with the Framework service registry and are notified with an Event object when an event is broadcast.

EventHandler objects can inspect the received Event object to determine its topic and properties.

EventHandler objects should be registered with a service property EventConstants.EVENT_TOPIC[p.124] whose value is the list of topics in which the event handler is interesed.

For example:

```
String[] topics = new String[] {"org/osgi/topic", "com/isv/
*"};
Hashtable ht = new Hashtable();
ht.put(EVENT_TOPIC, topics);
context.registerService(EventHandler.class.getName(), this,
ht);
```

If an EventHandler object is registered without a service property EventConstants.EVENT_TOPIC[p.124], then the EventHandler will receive events of all topics.

Security Considerations. Bundles wishing to monitor Event objects will require ServicePermission[EventHandler,REGISTER] to register an EventHandler service. The bundle must also have TopicPermission[topic, SUBSCRIBE] for the topic specified in the event in order to receive the event.

See Also Event[p.122]

7.11.5.1 public void handleEvent(Event event)

event The event that occurred.

- Called by the EventAdmin[p.123] service to notify the listener of an event.

7.11.6 public final class TopicPermission extends Permission

A bundle's authority to publish or subscribe to event on a topic.

A topic is a slash-separated string that defines a topic.

For example:

```
org/osgi/service/foo/FooEvent/ACTION
```

TopicPermission has two actions: PUBLISH and SUBSCRIBE.

7.11.6.1 public static final String PUBLISH = "publish"

The action string publish.

7.11.6.2 public static final String SUBSCRIBE = "subscribe"

The action string subscribe.

7.11.6.3 public TopicPermission(String name, String actions)

name Topic name.

actions PUBLISH,SUBSCRIBE (canonical order).

- Defines the authority to publich and/or subscribe to a topic within the EventAdmin service.

  The name is specified as a dot-separated string. Wildcards may be used. For example:

  ```
  org/osgi/service/fooFooEvent/ACTION
  com/isv/*
  *
  ```

  A bundle that needs to publish events on a topic must have the appropriate TopicPermission for that topic; similarly, a bundle that needs to subscribe to events on a topic must have the appropriate TopicPermssion for that topic.

7.11.6.4 public boolean equals(Object obj)

obj The object to test for equality with this TopicPermission object.

- Determines the equality of two TopicPermission objects. This method checks that specified Topic has the same Topic name and TopicPermission actions as this TopicPermission object.

Returns true if obj is a TopicPermission, and has the same Topic name and actions as this TopicPermission object; false otherwise.

7.11.6.5 public String getActions( )

- Returns the canonical string representation of the TopicPermission actions.

Always returns present TopicPermission actions in the following order: PUBLISH,SUBSCRIBE.

Returns Canonical string representation of the TopicPermission actions.

7.11.6.6 public int hashCode( )

- Returns the hash code value for this object.

Returns A hash code value for this object.

7.11.6.7 public boolean implies(Permission p)

p The target permission to interrogate.

- Determines if the specified permission is implied by this object.

  This method checks that the topic name of the target is implied by the topic name of this object. The list of TopicPermission actions must either match or allow for the list of the target object to imply the target TopicPermission action.

```
x/y/*,"publish" -> x/y/z,"publish" is true
*,"subscribe" -> x/y,"subscribe"   is true
*,"publish" -> x/y,"subscribe"     is false
x/y,"publish" -> x/y/z,"publish"   is false
```

Returns true if the specified TopicPermission action is implied by this object; false otherwise.

7.11.6.8 public PermissionCollection newPermissionCollection( )

- Returns a new PermissionCollection object suitable for storing TopicPermission objects.

Returns A new PermissionCollection object.

8 Deployment Admin Specification

Version 1.0

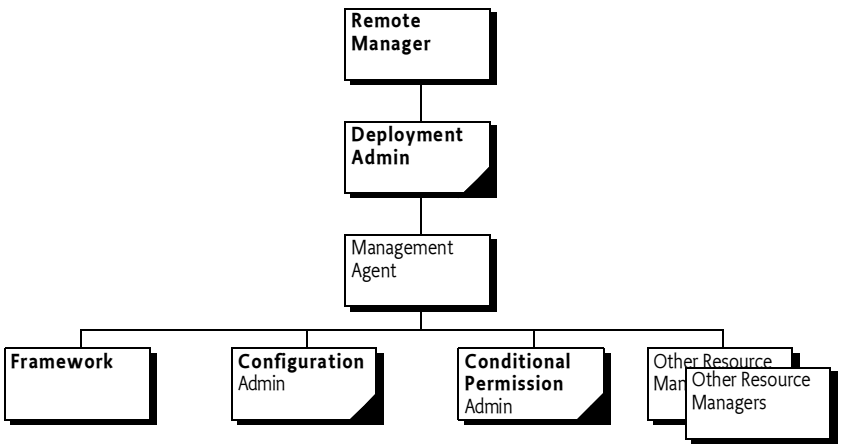
8.1 Introduction

The ability to install new software components after the time of manufacture is of increasing interest to manufacturers, operators and end users. For example, end users already are, or soon will be, accustomed to installing applications or services on their devices from remote servers.

The OSGi Service Platform provides mechanisms to manage the life cycle of bundles, configuration objects, and permission objects but the overall consistency of the runtime configuration is the responsibility of the *management agent.* In other words, the management agent decides to install, update, or uninstall bundles, create or delete configuration or permission objects, as well as manage other resource types.

The task of the management agent is extensive because it must track the, sometimes fine grained dependencies and constraints between the different resource types. This model, though extremely flexible, leaves many details up to the implementation. This significantly hinders the inter-operability of devices because management systems must supply their own management agents resulting in proliferation on the management server side. This specification therefore introduces the Deployment Admin service that standardizes deployment and management of interlinked resources to an OSGi Service Platform.The role of the Deployment Admin is depicted in Figure 25.

Figure 25 Deployment Admin role



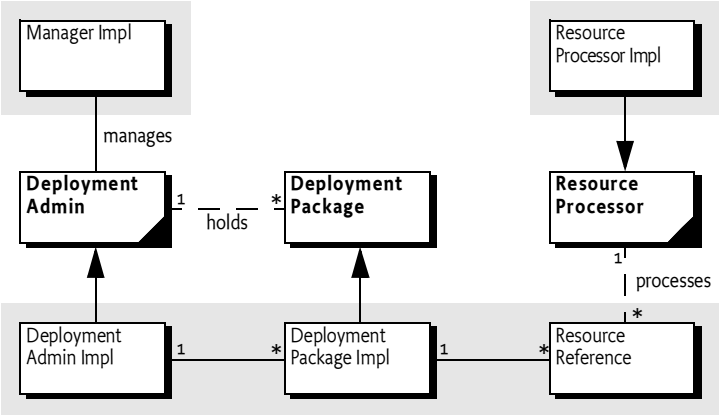
8.1.1 Essentials

TBD

8.1.2 Entities

- *Resource Type* – The general term from something that is life cycle managed by the Deployment Admin service. For example, bundles, configuration objects, and permissions are different resource types.
- *Deployment Admin Service* – The service that is used to install and uninstall Deployment Packages.
- *Resource Processor* – A service that can handle the life cycle of a specific resource type.
- *Deployment Package* – A group of resources that must be treated as a unit. I.e. there exists unbreakable dependencies between these resources.

Figure 26 *Deployment Admin Service, org.osgi.service.deploymentadmin package*



8.1.3 Synopsis

A developer can package a number of resources in a Deployment Package. A Deployment Package is stored in a JAR file, with format that is similar to bundles. A Deployment Package JAR can be installed via the Deployment Admin service. The Deployment Admin service manages the bundle resources itself but will process each resource in the deployment package by handing it off to a Resource Processor service that is designated for that resource type.

If all resources have been processed, the changes are committed. The Deployment Admin service is, however, not guaranteed to be fully transactional.

8.2 Deployment Package

A Deployment Package groups resources as a unit of management, something that can be installed, updated, and uninstalled as an atomic unit.

A deployment package is a set of related *resources* that need to be managed as a *unit* rather than individual pieces. For example, a Deployment Package can contain both a bundle as well as its configuration data.

It is *not* a plan from one consistent state to another; several deployment packages may be needed to achieve a new consistent state. Like bundles, a deployment package does not have to be self-contained unit. Its bundle resources can have dependencies on Java packages and services provided by other deployment packages.

For example, a suite of games shares some common parts. The suite contains two games: Chess (com.acme.chess) and Backgammon (com.acme.backg). Both share a top-score database as well as a 3D graphic library.

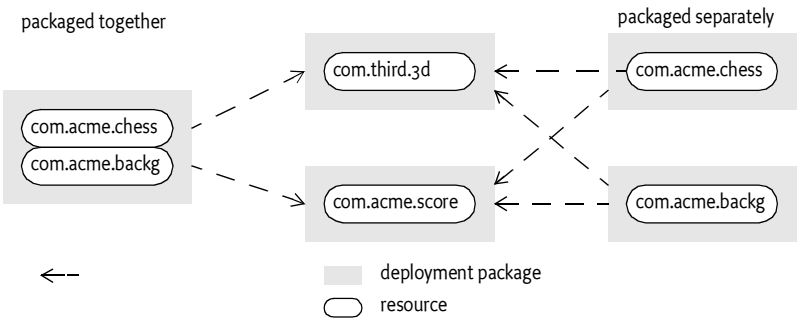
- com.third.3d – The 3D graphic library comes from a third-party provider. It is a deployment package of its own, composed of several bundles and possible configurations.
- com.acme.score – The top-score database would also be its own deployment package, and would in fact be optional. It offers a service for storing top scores, but games can function without this service.

Each game is a Deployment Package. This allows them to be installed independently. Alternatively, the two games can be packaged into the same deployment package, but in this case they must be installed and removed together and can no longer be deployed independently.

However, these two different packaging strategies are not compatible. Once the games are deployed separately, they can no longer be grouped later in an update because this would move ownership of the bundle to another deployment package and this is specifically not allowed. A bundle can only belong to one deployment package.

These scenarios are depicted in Figure 27.

Figure 27 Packaged game



Deployment packages are managed as first-class objects during run time, similarly to bundles. The DeploymentPackage interface represents this run time object.

A Deployment Package JAR can be created by developers but versioned and signed by a provider once it is tailored to its particular needs. It is also possible that deployment packages be created solely by a provider to effect certain changes onto a device.

8.2.1 Atomicity and Sharing

A Deployment Package is a reified concept, like a bundle, in an OSGi Service Platform. It is created and managed by the Deployment Admin service. As a unit, a deployment package should be installed, updated, or uninstalled atomically.

Deployment packages are essentially an ownership model for resources installed in an OSGi Service Platform. A deployment package assembles resources as a downloaded stream, which once processed, will result into the installation of a number of resources in the OSGi Platform such as:

- Installed bundles
- Configuration objects
- System properties
- Certificates
- Wiring schemas

A deployment package will *own* its resources. If a Deployment Package is uninstalled, all its resources must be uninstalled as well. The ownership model follows a *no-sharing* principle: *equal* resources are *not* shared between deployment packages.

The meaning of equal is dependent on the resource type. For example, two bundles are considered equal if their bundle symbolic name is equal, regardless of the version.

A sharing violation must be considered an error. The install or update of the offending deployment package must fail if a resource would be affected from another deployment package.

This is really only true for bundles unless we create global resource identifiers?

Do we need an exception for this?

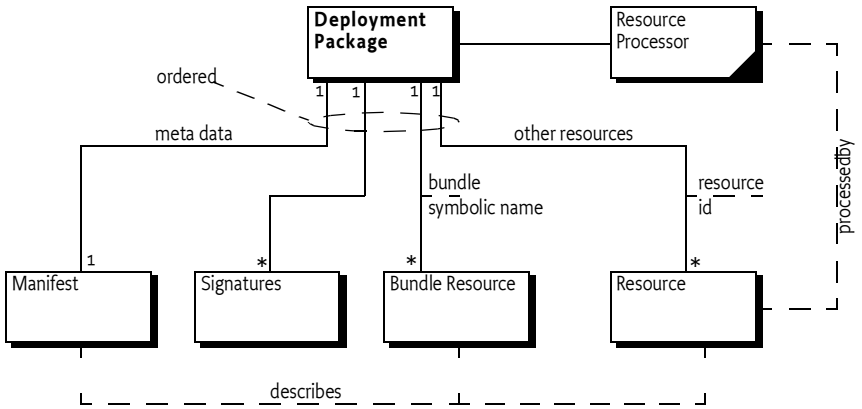
For example, a deployment packages could be used to install bundles and Configuration objects for Managed Services. Because of the no-sharing principle, an installed bundle must belong to one, and only one, deployment package (as defined by its bundle symbolic name). A configuration object (as defined by its PID) may belong to one and only one deployment package as well. Therefore, trying to install a deployment package where one of the bundle or one of the configuration objects is already present is an error and the install must fail.

Again, this is not completely true because PIDS are not really visible in the DP interface.

This strong no-sharing rule ensures a clean and robust life cycle. It allows the simple cleanup rule: the deployment package that installs a resource is the one that uninstalls it.

8.3 Structure of a Deployment Package

Figure 28 Structure of a Deployment Package



Do we have a resource id as shown?

8.4 File Format

A Deployment Package is a standard JAR file as specified in [15] *JAR File Specification*. The extension of a Deployment Package JAR file name must be .dp. E.g.valid names are:

```
com.acme.chess.dp
chess.dp
```

Should we ask for a MIME type at IANA?

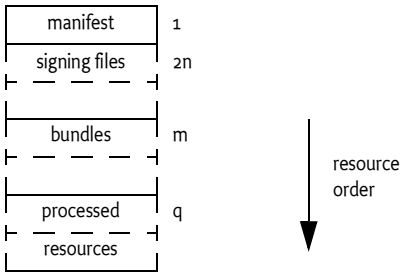
A Deployment Package must be formed in such a way that it can be read with a JarInputStream object, while supporting the getManifest() method as well as the resources used for signing of the entries. This requires that the order of the files in the JAR file defined to be:

- META-INF/MANIFEST.MF – A Deployment Package must begin with a standard Java manifest file. This rule is not explicit by the Java JAR file specification, however, it is implied by the known JarInputStream class implementations.
- META-INF/*.SF, META-INF/*.DSA, META-INF/*.RS – If the Deployment Package is signed, subsequent files in the JAR must be the signature files as defined in the manifest specification. The signature files are not considered resources.

do we support more than DSA/RSA?

- *Bundles* must come before any other resource types so that they can be installed before any other resources. Bundles must be installed and started in resource order on the JAR file and stopped in reverse order.
- *Resources* – Any other resources needed for this package. Resources are processed in the order of which they appear in the JAR file; and dropped in reverse order.

Figure 29 *Deployment Package JAR format*



The order of all the resources in the JAR file is significant, and is called the *resource order*. The purpose of the resource order is to allow the JAR to be processed as a stream. I.e. it is not necessary to buffer the input stream in memory, or to hard disk, to allow random access to its contents. All access to the stream must therefore be sequential. To increase the determinism, the resource order must also determine the processing order of the bundles and the resources.

The format is shown graphically in Figure 29.

Do we limit the name of the JAR path? Length, character set? The path name is an important key and it would be nice if it was not necessary to escape it in filter ...

8.4.1 Signing

Deployment packages are optionally signed by JAR signing, compatible with the operation of the standard java.util.jar.JarInputStream class, i.e. as defined in [15] *JAR File Specification*. This compatibility requires that the manifest must be the first file in the input stream, and the signature files must follow.

A deployment package can be signed by many signers. However, and this an OSGi unique requirement, *all* signers must sign *all* files in the JAR file except for the signature files. A partially signed deployment package JAR is not a valid deployment package and must be rejected.

8.4.2 Deployment Package Manifest

The manifest of a Deployment Package consists of a global section and separate sections for each resource contained within it, called the *Name section*. The global section can contain the following headers:

- DeploymentPackage-ManifestVersion – This header specifies the specification version of the manifest. For this specification it must be 1. See further *DeploymentPackage-ManifestVersion* on page 134.

why do we need this header? if we need it later, we can always introduce it. It just makes examples more messy. Seems completely unnecessary to me.

- DeploymentPackage-Name – The name of the deployment package as a reverse domain name. For example, com.acme.chess. See further *DeploymentPackage-Name* on page 134.

- DeploymentPackage-Version – The version of the deployment package as defined in [16] *OSGi Framework*. See further *DeploymentPackage-Version* on page 135.
- DeploymentPackage-FixPack – Marks this deployment package as a partial update to a resident deployment package. See *Fix Package* on page 141.
- DeploymentPackage-Processing – Provides a set of associations between resources and resource processors. See *Customizer* on page 140.
- DeploymentPackage-ProcessorBundle – Marks a bundle in the deployment package as a customizer. See *Matched Resource Processors* on page 139.

As with any Java manifest, additional headers can be added and must be ignored by the Deployment Admin service. If any fields have human readable contents, natural language translation may be provided through property files as described in [16] *OSGi Framework*. The Deployment Admin service must always use the raw, untranslated, version of the header values.

Where is the call to access these headers? How can I see any translations? Does localization for this file make sense? At least we need a list of headers where it does not apply for.

For example:

```
Manifest-Version: 1.0
DeploymentPackage-ManifestVersion: 1
DeploymentPackage-Name: com.third._3d
DeploymentPacakge-Version: 1.2.3.build22032005
↵
```

Should we make *-Name into *-SymbolicName? So that it is clear what is human readable and what not?

Don't we need headers for docurl, vendor, copyright, etc?

Additionally, the manifest must carry a Name section for each resource in the JAR file (except the signature files). Each name section must start with an empty line (carriage return and line feed shown as ↵).

The Name section must start with a Name header that contains the path name of the resource. The path name must not start with a slash. This is a requirement from the standard JAR Manifest. E.g.

```
Name: bundles/3dlib.jar
```

The name section can include relevant meta data for the named resource. For bundles, which are the default resource type for the Deployment Admin service, only the specification of the Bundle-SymbolicName and Bundle-Version headers are required. Unrecognized headers are allowed and must be ignored by the Deployment Admin service.

The following headers are architected for the Name section in the manifest of a deployment package:

- Bundle-SymbolicName – Only for bundle resources. This header must be identical to the bundle symbolic name of the named bundle.

Will this be verified after an install?

- Bundle-Version – Only for bundle resources. This header must be identical to the bundle version of the named bundle. Its syntax must follow the version syntax as defined in the Framework specification.
- MissingResource-Bundle –See *Fix Package* on page 141, a Bundle Symbolic Name for a bundle that is part of this deployment package but must already be installed on the device and is therefore not carried in this JAR file.
- MissingResource-Resource – See *Fix Package* on page 141, contains a resource identifier. The resource identifier is the only available identity for these resources and the interpretation depends on the resource type.

I am confused with this ID, it is a good idea to identify the resources in a global way to handle the sharing, but this header is a bit odd.

An example manifest of a deployment package that deploys the 3D package consisting of 2 bundles and no resources.

```
Manifest-Version: 1.0
DeploymentPackage-ManifestVersion: 1
DeploymentPackage-Name: com.third._3d
DeploymentPacakge-Version: 1.2.3.build22032005
DeploymentPackage-Processing:
  processor = "(service.id=org.osgi.autoconf)";
    resources = META-INF/autoconf.mf
↵
Name: bundles/3dlib.jar
SHA1-Digest: MOez1l4gXHBo8ycYdAxstK3UvEg=
Bundle-SymbolicName: com.third._3d
Bundle-Version: 2.3.1
↵
Name: bundles/3dnative.jar
Bundle-SymbolicName: com.third._3d.native
Bundle-Version: 1.5.3
SHA1-Digest: N8Ow2UY4yjnHZv5zeq2I1Uv/+uE=
↵
Name: META-INF/autoconf.xml
SHA1-Digest: M78w24912HgiZv5zeq2X1Uv-+uF=
↵
```

8.4.2.1 DeploymentPackage-ManifestVersion

Defines the grammar of the Deployment Admin headers version. For this specification, this version must be 1. The format is:

```
DeploymentPackage-ManifestVersion ::= '1'
```

If this header is absent, then the JAR is not a valid Deployment Package.

Propose to kill this header

8.4.2.2 DeploymentPackage-Name

The name of the deployment package. A name must follow the same rules as Java packages. The grammar is as follows:

```
DeploymentPackage-Name ::= unique-name
```

This is a mandatory header.

An example is:

```
DeploymentPackage-Name: com.acme.chess
```

8.4.2.3 DeploymentPackage-Version

This header defines the version of the deployment package. The syntax follows the standard OSGi Framework rules for versions.

```
DeploymentPackage-Version ::= version
```

This is an optional header, however, if it is specified it must follow the syntax of the version.

An example:

```
DeploymentPackage-Version: 1.2.3.build200501041230
```

8.4.2.4 DeploymentPackage-Processing

Operations carried out upon resources are processed by Resource Processor services. This requires the Deployment Admin service to know what resource is bound to what Resource Processor service. This association is therefore made with the help of this header.

The DeploymentPackage-Processing matches an OSGi Resource Processor service to a set of contained resources. The format of the header is:

```
DeploymentPackage-Processing ::= process ( ',' process ) *

process ::= attribute ( ';' attribute ) +
```

this syntax is yet different from the standard name+ (';' attribute) * syntax.

do we allow other attributes? Might be useful for the RP. The shown syntax allows this.

The following attributes are architected:

- process – (OSGi filter, see [16] *OSGi Framework*) Defines a filter that must select a Resource Processor service that will be allowed to process the resources as defined by the resource attribute. The service must be registered under the ResourceProcessor interface. I.e. this requirement must be automatically added to the filter. It is therefore not possible to select an arbitrary service with the filter.
- resource – (path) Defines the resources in this deployment package that must be processed by the selected Resource Processor service. The pathname in the resource filter is relative to the root directory of the deployment package and uses a simple wildcard mechanism to select the files in the deployment package. The following wildcards can be used:
  - * – Matches 0 or more occurrences of any character
  - ? – Matches 1 occurrence of any character

More OSGi like would be:

DeploymentPackage-Processing:= path (';' path) * (';' attribute) *

However, an even better solution might be to put the processor selection as a header on the Name section.

Name: certificates/thawte.x509

DeploymentPackage-Processor: (service.pid=2345678912.121)

This model makes it easier to merge bundles and delegates some processing and complexity to the deplore, where it should be. It is also easier to inspect, errors and intrusions are more obvious.Or better, skip the filter and specify a PID instead of a filter

For example, defining two processors to apply to two sets of files:

```
DeploymentPackage-Processing:
   processor="(service.pid=com.securitas.keystore)";
      resources = certificates/*.cer,
   processor="(service.pid=com.florence.help)";
      resources = *.hlp
```

A filter for the procesor seems a bit overkill. The PID would be easier and more consistent. It would also make it easier to detect overlaps.

Within the example, the certificates in the certificate directory will be processed by the Resource Processor service registered with the service property service.pid set to com.securitas.keystore. The service.pid is a standard Framework property to uniquely identify a service instance called a Persistent IDentity a.k.a. PID.

The second declaration in the example specifies that the Resource Processor service with PID com.daffy.help will process all resources in the root directory with the .hlp extension.

8.4.2.5 DeploymentPackage-ProcessorBundle

A resource processor can be delivered within a deployment package as a bundle. Such a resource processor is called a *customizer*. Customizer bundles are identified through the use of the manifest header DeploymentPackage-ProcessorBundle. The header has the following format:

```
DeploymentPackage-ProcessorBundle ::= unique-name
```

This limits to a single customizer, looks rather restrictive?

Could we not call this DeploymentPackage-Customizer?

For example, a resource processor contained within the deployment package, with a Bundle Symbolic Name of com.acme.chess.customizer could be identified by the following header:

```
DeploymentPackage-ProcessorBundle: com.acme.chess.customizer
```

Customizers are further explained at *Customizer* on page 140.

The use of a BSN opens up a lot of possible errors. This field should actually be a path, or better, it should be a header in the name section of the customizer.

8.4.2.6 DeploymentPackage-FixPack

A fix package is an optimized Deployment Package format that only contains the resources that are pertinent to an update relative to an installed Deployment Package. I.e. it lacks already installed resources to reduce the download size. The fix package assumes that a particular deployment package is already installed. It must specify the range of versions of deployment packages for which it is applicable.

A fix package can be distinguished from a full format through the presence of the DeploymentPackage-FixPack manifest header, contained within the global section of the manifest. The format of this header is:

```
DeploymentPackage-FixPack ::= version-range
```

The version range syntax is identical to the Framework module's layer version range as defined in [16] *OSGi Framework*. For example, a manifest header that denotes a fix package which is only applicable to versions 1.3 through 3.4, inclusive, of a given deployment package looks like:

```
DeploymentPackage-FixPack: [1.3,3.4]
```

The example was mismatched to the test, the R4 version range is OK?

8.4.2.7 Bundle-SymbolicName (Name Section)

The Bundle-SymbolicName header must be a copy of the Bundle-SymbolicName header in the named bundle, including any parameters. The header has the following format:

```
Bundle-SymbolicName: unique-name (';' parameter ) *
```

For example:

```
Name: bundles/http.jar
Bundle-SymbolicName: com.acme.http; singleton=true
```

See *Resources* on page 138 for more information about the use of this header.

8.4.2.8 Bundle-Version (Name Section)

The Bundle-Version header must be a copy of the Bundle-Version header in the named bundle. It must follow the format as defined version clause in [16] *OSGi Framework*.

```
Bundle-Version ::= version
```

For example

```
Bundle-Version: 1.2
```

See *Resources* on page 138 for more information about this header.

8.4.2.9 MissingResource-Bundle (Name Section)

Fix packs, see *Fix Package* on page 141, are deployment packages that miss a number of resources for a full install. This header defines the bundle symbolic name of a bundle that is not present in the enclosing JAR file but should be part of a prior version of this deployment package. The format is:

```
MissingResource-Bundle ::= unique-name ( ';' parameter )*
```

For example:

```
Name: bundles/3dlib.jar
MissingResource-Bundle: com.acme.http
Bundle-SymbolicName: com.acme.http
Bundle-Version: 3.0
```

This header really does not make sense. It looks much easier to explain to have a single header: DeploymentPackage-Missing: true / false. The rest of the information should then be the same as if the resource was there. This makes processing a lot easier and more consistent.

8.4.2.10 MissingResource-Resource (Name Section)

Fix packages, see *Fix Package* on page 141, are deployment packages that miss a number of resources for a full install. This header defines an identity for a resource that should be installed on the system but is not packaged in the enclosing JAR file. The format is:

```
MissingResource-Resource ::= unique-name
```

What is the purpose of the unique name, examples?

8.4.3 Deployment Package Naming

Every Deployment Package must have a name and version. Package authors should use unique names through the usage of reverse domain naming, i.e. like the naming used for Java packages. The version syntax must follow the rules defined in the Framework's Module layer. If the version is not specified in the JAR, 0.0.0 is assumed.

Together, the name and version specify a unique Deployment Package; a device will consider any Deployment Package with identical name and version pairs to be identical.

Deployment packages with the same name but with different versions are considered to be *versions* of the *same* deployment package. The Deployment Admin service maintains a set of installed deployment packages. This set must not contain multiple versions of the same deployment package. Installing a deployment package when a prior or later version was already present must replace the existing deployment package. This can be both an upgrade or downgrade.

8.4.4 Localization

TBD, until I understand what the scope of the headers is

8.5 Resources

A Deployment Package consists of installable *resources*. Resources are described in the name sections of the manifest and stored in the JAR file under a path.

A subset of these resources are the bundles. Bundles are treated different then the other resources by the Deployment Admin service. Non bundle resources are called *processed resources*.

Bundles are managed by the Deployment Admin service directly. The Deployment Admin service must set the bundle location of these bundles to their Bundle Symbolic Name, leaving out the parameters.

Processed resources are not managed directly by the Deployment Admin service but their management must be handed off to an appropriate Resource Processor service.

The installation and uninstallation processes are specified in detail in *Installing a Deployment Package* on page 143 and *Uninstalling a Deployment Package* on page 150.

8.6 Matched Resource Processors

Processed resources must be associated with a Resource Processor service. This association is made by the deployment package manifest with the DeploymentPackage-Processing header. This header defines a number of *processing jobs*. A processing job consists of a filter that must select a *single* Resource Processor service, and a path, with optional wildcard characters, that select a subset of the resources in the JAR file.

Therefore, using this information, some resources can be *matched* to a Resource Processor service. I.e. the matched Resource Processor service of a resource is the Resource Processor service that must process and later drop that resource according to the DeploymentPackage-Processing header.

To find the matched Resource Processor service of a resource, its path must be matched to the resources path of all processing jobs. Only one such job must match. The processing jobs Resource Processor service is then the matched Resource Processor of the resource.

This is done sooooo complicated... It would be a lot simpler if the name section just contained a PID instead of this global header and filter. There are now TOO many error cases: multiple services matched by the filter as well as multiple processing jobs matching a resource. n:m relations are hard ...

For example, assume the global section has specified the following DeploymentPackage-Processing header:

```
DeploymentPackage-Processing:
  processor = "(service.id=org.osgi.autoconf)";
    resources = META-INF/autoconf.mf,
  processor = "(service.id=com.acme.com.certificates);
    resources = certificates/*.cer
```

To find the matched Resource Processor of the resource META-INFO/autoconf.mf, the Deployment Admin service must find the matching processing job. This is the first clause, the matching Resource Processor is therefore the service identified by the filter (service.id=org.osgi.autoconf). It is an error when anything but one Resource Processor services matches this filter.

Resource Processors are discussed at *Resource Processors* on page 152.

8.7 Customizer

The standardized Deployment Admin service installation and uninstallation functions do not always cover the needs of a developer. In certain cases it is required to run some custom code at install and uninstall time. This is supported with the Deployment Package *customizer*. Typical customizers are:

- Database initialization
- Data Conversion
- Wiring

A customizer bundle (there can not be more than one) is specified in the DeploymentPackage-ProcessorBundle header.

Why is it only one?

A customizer specified in this manner must be part of the bundles in the deployment package. The customizer bundle must be installed and started by the Deployment Admin service *before* any of the resources are processed.

As the customizer bundle is started, it must register one ore more Resource Processor services in its activator start method. These Resource Processor services must only be used by resources originating from the same deployment package. I.e. customizer bundles must never process a resource from another Deployment Package.

what happens when it does not register an RP?

8.7.1 Bundle's Data File Area

The getDataFile on DP is a weird place ... could go to a session object

Each Bundle in the OSGi Framework has its own persistent storage area. This area is accessed by a bundle with the getDataFile method on the Bundle Context. The location in the file system where these files are stored is not defined and thus implementation dependent. However, a customizer typically needs access to this storage area.

The Deployment Admin service provides access to the Bundle private data area with the getDataFile(Bundle) method on the DeploymentPackage object.

I guess this method should be called getDataDir? or getDataRoot? The associated method on Bundle Context returns a specific file.

This method returns a File object to the root of the data directory.

How are we going to handle security???? The bundle gets this permission automatically, but the Rps not. The data file is "somewhere", so the RP requires access to all files, hardly secure. This is a potential maintenance nightmare. The java.io is broken, so the solution is horrible. This is a problem that has plagued us in many places, it requires a different approach to the File object. In java.io, the security check is done at stream creation, the File is therefore not capable of hiding the security issues.

I suggest to change the call (preferably on the session) to:

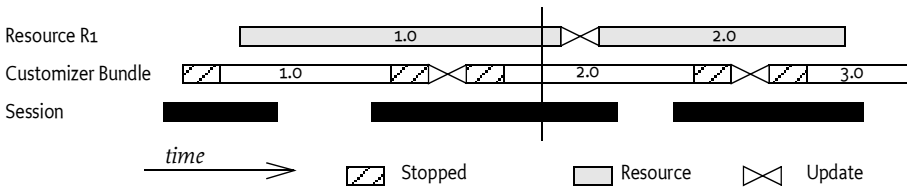
Directory getDirectory(Bundle)

```
interface Directory {
        InputStream getInputStream(String file);
        OutputStream getOutputStream(String file);
        String [] getFiles();
        Directory [] getDirectories();
        Directory getParent();
        RandomAccessFile getRandomAccessFile(String file);
}
```

8.7.2 Customizers and Update

The life cycle of a customizer bundle is intertwined with the life cycle of the resources it processes. Care should be take to ensure that updates and uninstallations are handled correctly. A customizer is updated *before* a resource is processed. This implies that a resource deployment session n is always *dropped* or *updated* by the customizer from session n+1. .

Figure 30 *Timeline for customizer versus resource versions*



In Figure 30, customizer 2.0 must update the resource from version 1.0 and customizer 3.0 must drop the resource from version 2.0. As a consequence, the customizer that installs a resource will be of a different version then the one that updates or uninstalls it.

Could we uninstall/stop the customizer? At least optionally? I.e. add a parameter on the header?

8.8 Fix Package

A fix package is a Deployment Package JAR that minimizes download time by excluding resources that are not required to upgrade an existing Deployent Package. It can only be installed on a Service Platform if a previous version of that Deployment Package is already installed. I.e. the fix package only contains the changed and new resources. A fix package (called the *source*) therefore must specify the range of versions that the existing Deployment Package (called the *target*) must have installed. This range is specified with the `DeploymentPackage-FixPack` header in the manifest of the source.

The manifest format for a fix package is the same as for a deployment package manifest: Each resource must be named in the name section of the manifest. However, resources that are excluded must be marked in the named section.

Thus, the name sections of the manifest of a fix package must list *all* resources, missing or present, in order to distinguish between resources that must be removed or resources that did not require an update.

Maybe trivial, why not just skip the contents and make everything equal. I.e. for a fix package it is OK when a resource is missing. For a non Fp, it is an error? And skip the headers ... This makes it very easy to automatically turn a DP into an FP ...

Two name section manifest headers are used for indicating the a resource is missing:

- MissingResource-Bundle – A Bundle Symbolic Name for a bundle that should alread be installed and is not carried in this JAR file. No specification of version is required, since only one version is allowed and a version range on the target has already been provided by DeploymentPackage-FixPack header in the source.
- MissingResource-Resource – Contains resource path. The resource path is the only available identity for these resources and the interpretation depends on the resource type.

don the this. Why do we have different headers? A bundle is a resource, albeit the default. What is the resource path? Can't we just mark the Name section with a missing marker?

Name: bundles/3dlib.jar

Bundle-SymbolicName: com.third.3d

Bundle-Version: 2.3.1

DeploymentPackage-Missing: true

Again, this looks like an unnecessary special case? The fix pack range is not easily related to the bundle's version. By making a fix pack the same as a normal dp, you make it a lot simpler and allow a lot of sanity checks as well as reuse the tools. E.g. if the full headers are available, the resource processors can verify that their state is aligned with the assumptions of the fix-pack.

Only a fix package (i.e. a deployment package that has the DeploymentPackage-FixPack header) is permitted to contain the missing resource headers.

For example, the following headers define a valid fix package.

```
Manifest-Version: 1.0
DeploymentPackage-ManifestVersion: 1
DeploymentPackage-FixPack: [1,2)
DeploymentPackage-Name: com.acme.package.chess
DeploymentPackage-Version: 2.1
↵
Name: chess.jar
```

```
Bundle-SymbolicName: com.acme.bundle.chess
Bundle-Version: 5.7
MissingResource-Bundle: com.acme.bundle.chess
↵
Name: score.jar
Bundle-SymbolicName: com.acme.bundle.chessscore
Bundle-Version: 5.7
↵
```

In the example, the fix package requires that version 1.x.y of the deployment package is installed. The com.acme.bundle.chess bundle is assumed to be present on the Service Platform and it must be part of the deployment package com.acme.package.chess. After installation this deployment package must contain the two bundles.

8.9 Installing a Deployment Package

Installation starts with the installDeploymentPackage(InputStream). There is no separate function for an update, if the given Deployment Package already exists it must replaced with this new version after installing new resources, updating existing resources and removing stale resources. The purpose of the installDeploymentPackage method is therefore to replace the target with the source.

The InputStream object must contain a valid Deployment Package JAR, it is called the *source* deployment package. It must be a general InputStream (not a JarInputStream) object. If there exists an installed Deployment Package with the same name as the source then it is called the *target* Deployment Package. If no target exists, an invisible empty target with a version of 0.0.0. must be assumed without any bundles an resources.

The installation of a deployment package requires the cooperation and interaction of a large number of services. This operation therefore takes place in a *session*. The implementation of the operations should make all effort to have transactional semantics for the session. The requirements on the transactionality are further described in *Aborting an Install* on page 147. In the following description, errors must abort the session unless specifically noted otherwise.

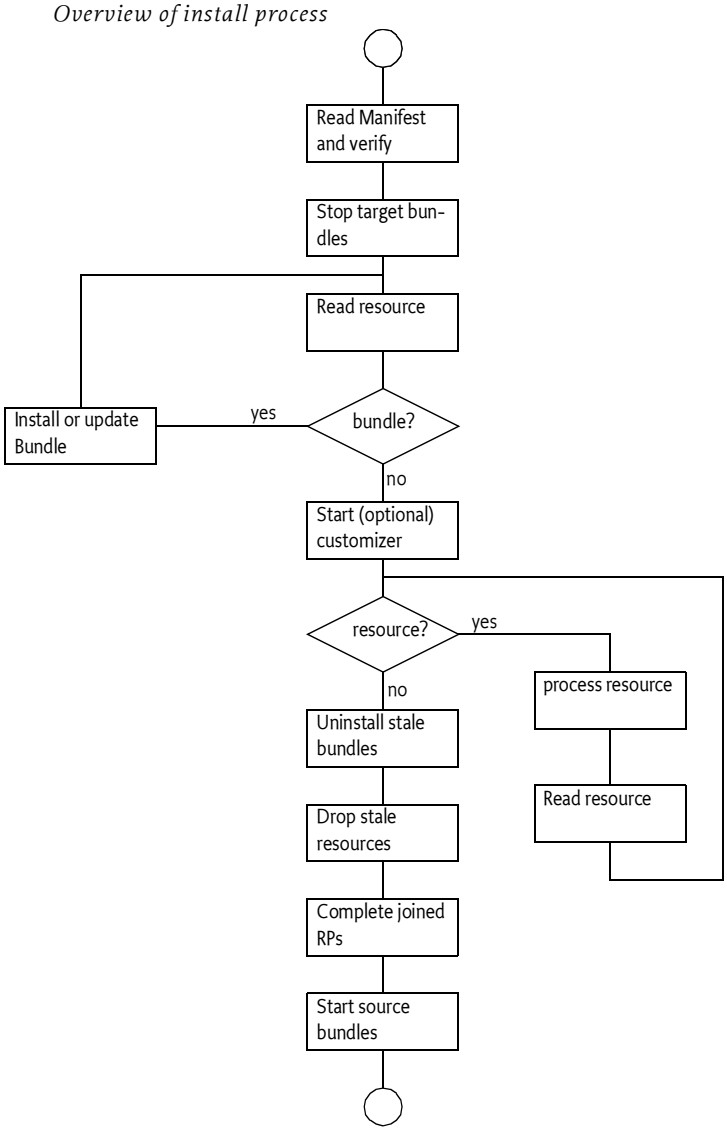
Resources must be processed by one, and only one, *matching* Resource Processor service. However, before using a resource processor in a session, the Deployment Admin service must *join* the session. The begin(DeploymentPackage,int) method must be called before a Resource Processor service is asked to process or drop a resource. For brevity, this joining is not shown in the detailed steps later but must be assumed to have taken place. The join must happen anytime during the session before the Resource Processor is used to process or drop a resource. A Resource Processor has *joined the session* when it has returned from its begin method without an Exception being thrown. A Resource Processor service must not be joined with more than a single session at any moment in time.

The installation of a deployment package can result in one of the types of qualifications for resources, for any resource r:

- $r \in source, r \notin target$ – New resource
- $r \notin source, r \in target$ – Stale resource
- $r \in source, r \in target$ – Updated resource

The short scenario for an install is depicted in Overview of install process-Figure 31.

Figure 31 *Overview of install process*



In more detail, to install a Deployment Package, a Deployment Admin service must:

1 Assert that the Manifest file is the first file in the JAR.
2 Read the manifest.
3 Assert the following:

- The source must not contain any bundle that exist in other deployment packages except for the target. I.e. the source bundles, as defined by their symbolic name, must belong to the target or be absent.

How is a bundle counted that is installed but does not belong to any DP?

Can a DP be downgraded?

If the source is a fix package, assert that:
- The version of the target matches the required source version range.
- All the missing source bundles are present in the target.

Any version checking here?

- All the missing source resources are present in the target.

Else, if the source is not a fix package:
- Assert that are no missing resources or bundles declared.

What do we do with version here? You would expect that the Bundle-Version would be a version range? Should the version match?

What happens when the bundle is in the target but it is not installed? Or it is not ACTIVE? Any verification here?

Are there more rules we need to check ahead of time?

4 All target bundles must be stopped in reverse target resource order. Exceptions thrown during stopping must be ignored but should be logged as a warning.

What if the bundles are not running?

The target is now stopped, none of its bundles are running anymore. The next part requires the sequential processing of the resources from the source JAR file in source resource order. The bundles must be first (if present) and must then be followed by zero or more resources.

For each bundle read from the source JAR stream:

5 If the bundle symbolic name of the bundle already exists in the system with a different version, then update that bundle with the resource stream. The `update` method must follow the semantics of the OSGi Framework `update` method. An exception thrown during the update must abort the session.

What do we do when the version is the same? One strategy is to update it anyway so you can fix a broken package. On the other hand, it is a lot of work for something that can also be achieved in another way: uninstall the DP and reinstall. I suggest we skip it, though the manifest bsn and version seem to be a bit feeble to make that decision on, but that is what is decided.

Else, install the bundle according to the semantics of the OSGi Framework `installBundle` method.The location of the bundle must be set to the Bundle Symbolic Name without any parameters. An exception thrown during the install must abort the session.

6 Assert that the installed bundle has the Bundle Symbolic Name as defined by the source manifest.

Do we guarantee Framework events? If so when?

All the source's bundles are now installed or updated successfully. Next, the optional customizer must be started so that it can participate in the resource processing of the resources:

7 If a customizer bundle is defined, start it. If this throws an exception, the session must be aborted.
8 Assert that the customizer has registered one or more Resource Processor services.

What happens when they are not there?

For each resource read from the JAR stream:

9 Find one matching Resource Processor service. If this fails, the session must abort.
10 Assert that the matched Resource Processor service is not a customizer from another Deployment Package.

Why do we distinguish between INSTALL and UPDATE? For integrity reasons, it looks more reliable when the rp figures this out depending on its state.

11 Call the matched Resource Processor service process(String,InputStream) method. The argument is the JAR path of the resource. Any exceptions thrown during this method must abort the installation.

All resource updates and installs have now occurred. The next steps must remove any stale resources. First the bundles are uninstalled and then later the resources are dropped. Exceptions are ignored in this phase to allow repairs to always succeed, even if the existing pacakge is corrupted.

12 Uninstall all stale bundles in reverse target order, using the OSGi Framework uninstall method semantics. Any exceptions thrown should be logged as a warning but must be ignored.
13 Uninstall all the resource, in reverse target order, that are in the target but not in the source by calling the matching Resource Processor service dropped(String) method. Any exceptions thrown during this method should be logged but must be ignored.

The next step is to cleanup an resource processors that are no longer needed by the source. I.e. any target resource processors that are not in the source.

For each of the resource processor that are referenced in the target but no longer in the source:

This seems a rather unnecessary step? The RP should already know that it no longer has an association?

14 Assert that the Resource Processor service has been associated with the source, this must already be the case. If the RP was associated with the target, it implies there was a resource of that type. This must have been dropped during this session, meaning the association must have taken place.
15 Drop the resource processor by calling the Resource Processor service dropped() method. Any exceptions thrown during this method should be logged but must be ignored.

The deployment package is now cleaned up and can be activated and committed.

16 All the Resource Processor service's that have joined the session must now be committed. This achieved by calling the complete(boolean) method. The boolean argument must be true to indicate success. The order in which the Resource Processors are called must be the reverse order of joining. Any exceptions should be logged as a warning but must be ignored.
17 Refresh the Framework so that any new packages are are resolved.
18 Wait until the refresh is finished.

That is not nice, but we have to wait before we start the bundles ...

19 Start the bundles in the source resource order. Exceptions thrown during the start must abort the deployment operation.

The order of the last two is tricky. If the bundles start before the commit, they might not see the resources. However, if they start after the commit, we cannot rollback anymore ... what order do we choose, we seem to be screwed anyway.

The session is closed and the source replaces the target.

The installDeploymentPackage method returns the source Deployment Package object.

8.9.1 Aborting an Install

An abort can take place at any moment during the installation session. At the moment of the abort, a number of Resource Processor services can have joined the session. For each of these joined Resource Processor services, the Deployment Admin service must call complete(boolean) with a false as argument. The Resource Processor services must be called in the reverse order of joining.

The system should make every attempt to rollback the situation as it was before the session started:

- Updated resources must be restored to their prior state
- New resources must be uninstalled
- Stale resources must be reinstalled.

If the state can be restored succesfully, the target bundles must be refreshed and started again (if they were started before) before the method returns. If the rollback is not complete, the target bundles must not be restarted. This is to prevent running bundles with incompatible versions. An appropriate warning should be logged in this case.

The Deployment Admin service must uninstall any new bundles, install stale bundles and should rollback updated bundles. Rolling back a bundle update as well as reinstalling a stale bundle requires an implementation dependent back door into the OSGi Framework because the Framework specification is not transactional over multiple life cycle operations.

This specification, unfortunately due to memory constraints on embedded devices, does not mandate full transactional behavior. However, after an abort a Deployment Package must still be removable with all its resources and bundles dropped. I.e. an abort must not bring the Deployment Package in a state where it can no longer be removed or where resources become orphaned.

I.e. a compliant Deployment Admin service can create resource states that are invalid, or Deployment Packages that own invalid resources, after an aborted session. However, a Deployment Admin service must ensure that there are never orphaned resources.

This is a famous Windows problem! Agree with this text?

I think we need a state on the Deployment Package and maybe a method “repair”. We need to differentiate between the following states:

INSTALLING

INSTALLED

UNINSTALLING

UNINSTALLED

INVALID

8.9.2 Example Installation

The target Deployment Package has the following manifest:

```
Manifest-Version: 1.0
DeploymentPackage-ManifestVersion: 1
DeploymentPackage-Name: com.acme.daffy
DeploymentPackage-Version: 1
DeploymentPackage-Processing:
  processor=(service.id=RP-x); resource=*.x,
  processor=(service.id=RP-y); resource=*.y
↵
Name: bundle-1.jar
Bundle-SymbolicName: com.acme.1
Bundle-Version: 5.7↵↵
Name: r0.x↵↵
Name: r1.x↵↵
Name: r1.y↵↵
```

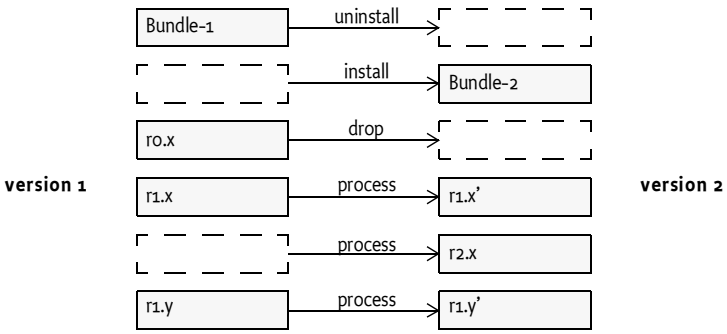
This deployment package is updated with a new version, with the following manifest:

```
Manifest-Version: 1.0
DeploymentPackage-ManifestVersion: 1
DeploymentPackage-Name: com.acme.daffy
DeploymentPackage-Version: 2
DeploymentPackage-Processing:
  processor=(service.id=RP-x); resource=r1.x,
  processor=(service.id=RP-x); resource=r2.x,
  processor=(service.id=RP-y); resource=r1.y
```

```
↵
Name: bundle-2.jar
Bundle-SymbolicName: com.acme.2
Bundle-Version: 5.7↵↵
Name: r1.x↵↵
Name: r2.x↵↵
Name: r1.y↵↵
```

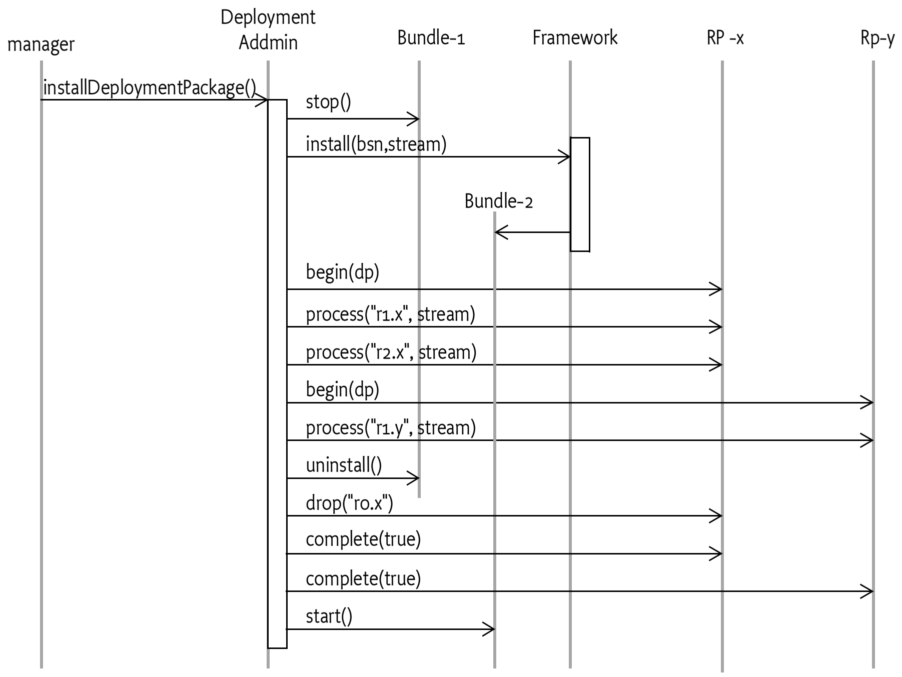
The delta between version 1 and version 2 of the com.acme.daffy deployment package is depicted in Figure 32. Bundle-1 must be uninstalled because it is no longer present in version 2. Bundle-2 is a new bundle and must thus be installed. the resource r0.x must be dropped, r1.x must be updated (this must be detected and treated accordingly by RP-x), and r2.x is a new resource. The RP-y resource r1.y is updated.

Figure 32 *Delta*



The sequence diagram for the installation is shown in Figure 33.

Figure 33 Sequence Diagram for a Resource Processor



###VERIFY~!!

8.10 Uninstalling a Deployment Package

Uninstalling a deployment package must have the effect of removing all the effects of its installation(s). The uninstall is triggered by calling uninstall() method on a *target* DeploymentPackage object.

The deployment packages are uninstalled explicitly, which may break the overall runtime configuration. There is no attempt to ensure that the uninstalled deployment package is not necessary as providers of Java packages or services. The uninstalling semantic for a deployment package is equivalent to the uninstalling semantic for a single bundle: it is mandatory, applies to all resources owned by the deployment package, and may break runtime dependencies.

The Deployment Admin service must take the following actions to remove the *target* Deployment Package.

How to get rid of a package? The RFC did not have an API for this. I have changed (well defined) the algorithm to always succeed. It looks like the uninstall requires a boolean parameter?

1 Assert that all Resource Processor associated with the target are present. If not, throw an ###Exception. (### should be conditional on a flag)

From this point on, the uninstall must always succeed.

2 Stop all the target bundles. Exceptions thrown should be logged as a warning but must be ignored.
3 Uninstall all target bundles. Exceptions thrown should be logged as a warning but must be ignored.
4 Drop all the target resources by calling the dropped(String) method on the matched Resource Processor service. If this fails, log this as a warning but continue.

It would be much more reliable if we could call RP.dropped(DP). Then we can be sure that the resources are dropped and no orphans remain. It also means we do not require a session???

Not finished

8.11 Introspection

The Deployment Admin service can provide the list of currently installed Deployment Packages with the listDeploymentPackages().

The method returns a list of DeploymentPackage objects. These objects provide access to:

- getId() – This must be an id that is assigned to the Deployment Package by the Deployment Admin. It must be a number that is increasing over time and it must never be reused. This number is not changed when the deployment package is updated with another version.
- getName() – The name of the deployment package. This name can be localized and is therefore not guaranteed to be unique. For localization see ###.
- getVersion() – The version of the Deployment Package.
- listBundles() – Return a list of all bundles that are *owned* by this Deployment Package.
- uninstall() – Uninstall this DeploymentPackage object. All methods must throw an IllegalStateException after the uninstall method returns.

Dont we need an installaton date or changed flag?

Don't we need a state for this object?

What happens when the object is uninstalled?

Dont we need access to the headers of the DP? It is usually a good idea in the UI to show the description, copyright etc.

Don't we need access to the resource list?

Don't we need to know what RPs are associated with this object?

The list of Deployment Packages must only contain installed packages that are valid. During an installation of an existing package, the target must remain in this list until the process is completed, after which the source replaces the target. I.e. if the installation fails, the source must never become visible, also not transiently.

The DeploymentPackage class must implement a hashCode and equals() so that equality is defined. Equality == same name? Identity == same package?

8.12 Resource Processors

The method names seem to be confusing because they do install/uninstall but are called process/drop. Why not call them install/uninstall? This is what a programmer would expect.

The API is not reflective. There is no way to find out what resources a resource processor holds. This is not the normal OSGi style.

The sessions is required for process/drop, but I wonder if it is needed for dropped()

The headers are not available to the RP, this seems very important

The Resource Processor service processes the byte stream of a processed resource. It is up to the Resource Processor service how to interpret this byte stream. Typically, the stream is parsed and its information is stored in a processor dependent place. Examples of resource processors are:

- *Configuration Management* – This processor is standardized by the OSGi and more information can be found in ###
- *Certificate Keystore* – A Certificate Keystore processor could extract certificates from a bundle and install these in a keystore.
- *SyncML Script* – Run a script in SyncML format.

The Deployment Admin service maintains the list of resource names (the path name in the JAR). Each resource is therefore uniquely identified within a deployment package.

This is not a global identifier so that makes it very difficult to do any checking of the resource ownership.

However, the Deployment Admin service must inform the Resource Processor about the resources that it is matching. The Resource Processor service is responsible for actually creating and deleting the related objects. Resources are identified with their JAR path. The Resource Processor service must be able to remove a resource that was installed earlier through its JAR path.

The ResourceProcessor interface is semi transactional. The interface is not fully transactional due to size constraints on embedded devices. The transactionality is limited to the bracketing of any processing or dropping of resources.The bracketing begins when a Resource Processor joins an install session, see *Installing a Deployment Package* on page 143.

Before the Resource Processor service is used in an install session, the Deployment Admin service must call the begin(DeploymentPackage,int) method. This method must be used by the Resource Processor service to mark any changes from now on until the complete(boolean) method is called. If the boolean argument is true, the marked changes should be persisted. If the argument is false, the marked changes should be undone. If the Resource Processor fails to undo the marked changes it must throw an ###Exception.

I would like to limit the session to the install... I am assuming this...

The Deployment Admin must not join a specific Resource Processor in more than one session. The Resource Processor service can assume that it will not be used with another session between the begin and matching complete method. I.e. a Resource Processor can ignore threading an re-entrancy from the Deployment Admin service issues. Nor can an open session overlap in time with the dropping of a Deployment Package.

When the session is opened, the Deployment Admin service can call the following methods:

- process(String,InputStream) – The Resource processor must parse the Input Stream and persistently associate the resulting objects with the given ID. I.e. it must be possible to remove those objects in a future time, potentially after a complete system restart. The method must be *idempotent*. I.e. calling this method more than once with the same resource for a given Deployment Packaage and resource path must have the same effect as calling it once.

I think this is an important requirement to increase reliability and it eases recovery, but it can make RPs harder to make.

- dropped(String) – The objects that were associated with the given ID must be removed. If the named resource does not exist, a warning should be logged but no exception must be thrown.

yes?

Both methods must perform any integrity checks immediately and throw and ###Exception if the verification fails. These checks must not be delayed until the complete method. As stated earlier, changes must be recorded but it should be possible to roll back the changes.

Deployment Packages can be upgraded, replaced and downgraded. Resource Processor services must therefore be capable of processing resources that are of a lower version than currently installed do not exist yet, or have a higher version.

8.12.1 Dropping Deployment Packages

The Deployment Admin service can ask the Resource Processor service to drop all information it has about a certain package with the dropped() method.

The Deployment Admin service must call this method when a Deployment Package is uninstalled or when the last resource associated with a specific Resource Processor is uninstalled.

I have two problems with this methd. First I think it should not be part of a session. This makes the implementation easier and reduces memory requirements. Second, it should not be the responsibility of the DA service to call it when the services are no longer required.

8.12.2 Example Resource Processor

For toy example, a Resource Processor service that wires services with the Wire Admin service. This service creates wires between a *producer* and a *consumer* service, each identified by a PID. For simplicity, this relation is stored in a format compatible with the java.util.Properties format. I.e. the key is the producer and the value is the consumer. A sample wiring could look like:

```
com.bunny.navigation = com.acme.gps.garmin
com.bunny.poi = com.acme.gps.garmin =
com.acme.gps.garmin = com.acme.debugger
```

This file is stored in a Deployment Package JAR file. In this example, there are no bundles so the Deployment Package's manifest would look like:

```
Manifest-Version: 1.0
DeploymentPackage-ManifestVersion: 1
DeploymentPackage-Name: com.acme.model.E45.wiring
DeploymentPackage-Version: 1.2832
DeploymentPackage-Processing: processor=
    "(service.id=wire.admin.processor)";
    resource="*.wiring"
↵
Name: sample.wiring
↵
```

To reduce the size of the code, the example is a class that gets the Wire Admin service as a parameter. The constructor registers the object as a Resource Processor service with the required wire.admin.processor PID.

The transaction strategy of this code is to create wires when new wires are found but delay the deletion of wires to the completed method. It tracks the created wires in the createdWires field and the to be deleted wires in the toBeDeletedWires field. Creation requires the current DeploymentPackage object so this is saved in the current field when the begin method is called.

This makes the first part of the class look like:

```
public class WireAdminProcessor implements ResourceProcessor
{
   WireAdmin          admin;
   DeploymentPackage  current;
   List               createdWires= new Vector();
   List               toBeDeletedWires= new Vector();

   public WireAdminProcessor(
        WireAdmin admin, BundleContext context)
        throws Exception {
      this.admin = admin;
      Dictionary properties = new Hashtable();
      properties.put(Constants.SERVICE_PID,
           "wire.admin.processor");
      context.registerService(
         ResourceProcessor.class.getName(), this,
         properties);
```

```
}
```

When the Deployment Admin service is installing a Deployment Package JAR, it must call the Resource Processor service's begin method. The given DeploymentPackage object is saved in the current field. The operation parameter is ignored.

Who needs the operation????

```
public void begin(DeploymentPackage dp, int operation) {
   current = dp;
}
```

The most complicated method that must be implemented is the process method. This method receives the path of the resource in the JAR and an Input Stream with its contents. In this case, the stream is easily converted to a java.util.Properties object.

The key and value of the Properties object are the producer and consumer respectively. They are used to create new wires. Each wire has a Dictionary object. Two fields are to the wire so it can be identified easily later:

- deployment.package – The name of the current deployment package.
- resource.id – The resource id, or JAR path name.

After a wire is created it is stored in the createdWires list so that it can be deleted if the session is aborted.

The method looks as follows:

```
public void process(String name, InputStream in)
        throws Exception {
   Properties properties = new Properties();
   properties.load(in);
   Dictionary dict = new Hashtable();
   dict.put("deployment.package", current.getName());
   for (Iterator i = properties.values().iterator();
        i.hasNext();) {
      dict.put("resource.id", name );
      String producer = (String) i.next();
      String consumer = properties.getProperty(producer);
      Wire wire = admin.createWire(producer,
         consumer, dict);
      createdWires.add(wire);
   }
}
```

If a resource is not in the source but it is in the target Deployment Package, it must be dropped from the Resource Processor service. The Wire Admin has a convenient function to get all the wires that match a filter. This method is used to list all the wires that belong to the current Deployment Package as well have the matching resource id. This array is added to the toBeDeletedWires field so that it can be deleted when the session is successfully completed.

```
public void dropped(String name) throws Exception {
   List list = getWires(
      "(&(resource.id=" + name + ")(deployment.package="
```

```
            + current.getName() + "))");
      toBeDeletedWires.add(list);
   }
```

The session complete method is now very simple. A choice must be made between deleting the wires in the `toBeDeletedWires` list if successful, or the wires in `createdWires` list if aborted.

```
   public void complete(boolean commit) {
      delete(commit ? toBeDeletedWires : createdWires);
      toBeDeletedWires.clear();
      createdWires.clear();
   }
```

The `dropped` method must drop all the wires that were created on behalf of the current Deployment Package. The filter on the `getWires` method makes this very straightforward.

```
   public void dropped() {
      List l = getWires("(deployment.package=" +
current.getName() + ")");
      delete(l);

   }
```

And at last, some helper methods that should be self explaining.

```
   void delete(List wires) {
      while ( ! wires.isEmpty() )
         admin.deleteWire((Wire) wires.remove(0));
   }

   List getWires(String filter) {
      try {
         Wire[] wires = admin.getWires(filter);
         return Arrays.asList(wires);
      }
      catch (InvalidSyntaxException ise) {
         ise.printStackTrace();
      }
      return new Vector();
   }
}
```

This example is obviously not an industrial strength implementation, its only purpose is to highlight the different problems that must be addressed. Implementers should consider the following issues when implementing a Resource Processor service.

- Changes could have been made to the Deployment Package objects when a Resource Processor's bundle is updated or has been offline for some time. The Deployment Admin service can provide sufficient information to verify its repository to the information maintained in the Resource Processor service.
- A Resource Processor should have a strategy for transactions that can handle recovery. For example, in the previous code the list of

createdWires and toBeDeletedWires should have been logged. This would have allowed full crash recovery.
- A better file format. The Properties class is too restrictive because it can only have a single wire per Producer object.

more tips?

8.13 Events

likely we need some events?

8.14 Threading

The Deployment Admin service is a singleton and must only process a single session at a time. When a client requests a new session with an install or uninstall operation, it must block that call until the earlier session is completed. The Deployment Admin service must throw an ###Exception when the session is can not be created after an appropriate time out period.

The introspection method can be called at any moment from any thread.

Resource Processor services can assume that all calls from begin to complete are all called from the same thread.

8.15 Security

8.15.1 Deployment Admin Permission

I am missing a permission.

8.15.2 Permissions During an Install Session

Unprotected, Resource Processor services can unwittingly disrupt the device by processing incorrect or malicious resources in a Deployment Package. In order to protect the device, Resource Processor service's capabilities must be limited by the permissions granted to the *signers* of a deployment package. This is called the *security scope*. If the Deployment Package JAR file is signed by multiple signers, the security scope is the union of the permissions of all signers. Given a signer, its permissions can be obtained from the Conditional Permission Admin service.###

The Deployment Admin service must execute all its operations, including calls for handling bundles and all calls that are forwarded to a Resource Processor, inside a doPrivileged block. This privileged block must use an AccessControlContext object that limits the permissions to the security scope. Therefore, a Resource Processor must assume that it is always running inside the correct security scope. This model can fail when the Resource Processor would do a doPrivileged itself, albeit this can sometimes be necessary.

8.15.2.1 Security Example

For example, the ACME cooperation is a GSM virtual operator. It is responsible for the management of its mobile phones but permits the Bugs company to deploy and manage its applications via its web site: http://wap.acme.com. I.e. The Bugs company can FTP deployment packages to the ACME wap site to be downloaded by their customers. ACME adds its signature to the Bugs application so that it can manage them as well without requiring overly broad permissions.

ACME has strict security requirements that Bugs can configure its own applications but it must never be allowed to configure other applications. To enforce this, ACME requires Bugs to sign its deployment packages. ACME therefore provides a signing certificate to Bugs.

There are therefore the following signing certificates in place:

- cn=ACME, o=Manager – The manager certificate is used to sign management applications or deployment packages. Managers must be able to configure all apps signed by ACME.
- cn=ACME, o=3p – The 3p certificate is used to sign third party apps before they are delivered to the phone. There are no permissions associated with this certificate, it is used so that the manager can use bundles that are signed with this certificate as a target.
- cn=Bugs, o=Manager – This is the signer of management applications. In this model, bundles signed with this certificate can set and get configurations of bundles signed by Bugs.
- cn=Bugs, o=Games – The games certificate is used to sign the applications of Bugs that must run in a sandbox.

The permissions, related to deployment, of these software providers could look as follows:

Configuration Permission does not work for this model. It is wrongly designed. I have adapted the example to fit the way it should be imho and have contacted Ben Reed.. I.e. the Configuration Permission must take a signer as target like Admin Permission. This require changes to the implementations though because this can only be checked when the PID is registered. or Admin Permission should handle configuration. However, if this change is not effectuated, the following example cannot be done and I do not know how to replace it with an example using the current design

It almost looks like we need a permission file in a deployment package...

Allow the ACME manager to get and set configurations for all bundles signed by ACME. Deployment Packages from Bugs are also signed by ACME so they can also be configured by ACME. The distinguished name (DN) is cn=ACME, o=manage:

```
Am:
ConfigurationPermission(
   "(signer=cn=ACME, o=*)", "get,set" ),
...
```

ACME will sign all third party products with the cn=ACME, o=3pp certificate (A3). This certificate has no permissions associated with, it is only marker.

Games from ACME must only be allowed to get their configuration from themselves or of other ACME games. Thus, permissions when signed by cn=ACME, o=game:

```
Ag:
ConfigurationPermission(
   "(signer=cn=Bugs, o=game)", "get" )
```

Allow a manager from Bugs to configure its own bundles, but nothing more. Permissions when signed by cn=Bugs, o=manager (Bm):

```
ConfigurationPermission(
   "(signer=cn=Bugs, o=*)", "*" )
...
```

Games from Permissions when signed by cn=Bugs, o=game:

```
Bg:
ConfigurationPermission(
   "(signer=cn=Bugs, o=game)", "get" )
...
```

The Configuration Resource Processor must be able to work for all possible bundles. It must therefore get full permission to the Configuration Admin.

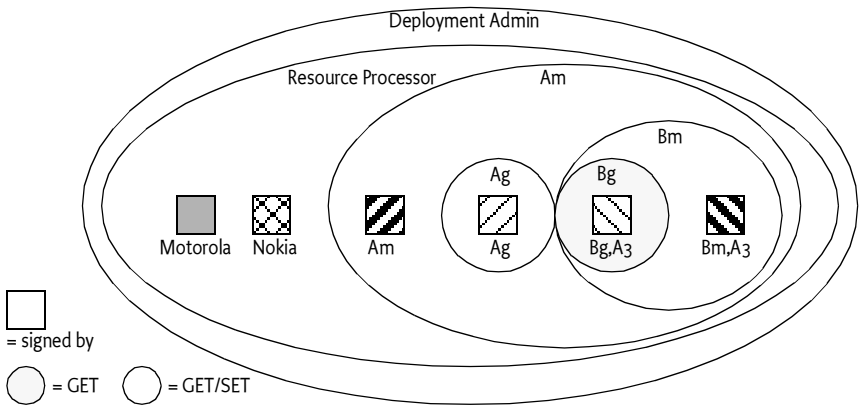
```
Resource Processor:
ConfigurationPermission("*","*"),
...
```

And last, the Deployment Admin service's permission also count:

```
Deployment Admin:
AllPermission()
```

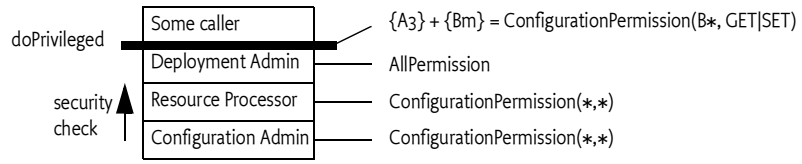
Graphically, this looks like Figure 34.

Figure 34 *Configuration permissions. B=Bugs, A=ACME, m=manager, g=game*



If the user downloads a Deployment Package with a game from Bugs, the package is signed by cn=ACME, o=3pp (A3) as well as cn=Bugs, o=manager (Bg). When the Configuration Resource Processor service executes an operation in the Configuration Admin service, the stack looks like Figure 35.

Figure 35 *Security stack for a deployment package.*



Any configuration data for the game can now be successfully set by the Resource Processor because it has sufficient permissions.

At first sight, it may look odd that a deployment package for a game requires it to be signed by a manager. However, deploying is a management operation. It is more secure if the actual games do not require the permission to install anything. As a consequence, a deployment package typically has *more* permissions than its constituent bundles.

8.15.3 Contained Bundle Permissions

Bundles can be signed independently from the vehicle that deployed them. As a consequence, a bundle can be granted more permissions than its parent Deployment Package.

For example, using the tables from the previous section, a Deployment Package signed by Bugs can be used to deliver a powerful management application signed by ACME.

This is the intention isn't it? Bundles are not automatically limited by their DP?

8.15.4 Service Registry Security

8.15.4.1 Deployment Admin Service

The Deployment Admin service is likely to require All Permission. This is caused by the plugin model. Any permission required by any of the Resource Processor service must be granted to the Deployment Admin service as well. This is a large and hard to define set. However, the following list shows the minimum permissions required if the Resource Processor service permissions are left out.

```
ServicePermission    ..DeploymentAdmin          REGISTER
ServicePermission    ..ResourceProcessor        GET
PackagePermission    org.osgi.service.deploymentEXPORT
```

8.15.4.2 Resource Processor

```
ServicePermission    ..DeploymentAdmin          GET
ServicePermission    ..ResourceProcessor        REGISTER
PackagePermission    org.osgi.service.deploymentIMPORT
```

8.15.4.3 Client

```
ServicePermission    ..DeploymentAdmin          GET
PackagePermission    org.osgi.service.deploymentIMPORT
```

Additionally, the local manager requires all permissions that are needed by the plugins it addresses.

8.16 org.osgi.service.deploymentadmin

Unexpected tag, assume para (x=39,y=8) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework's install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission("‹filter›","listDeploymentPackages") ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission("‹filter›", "installDeploymentPackage") ‹/blockquote› A holder of this permission can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission("‹filter›","uninstall") ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=25,y=12) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework's install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission("‹filter›","listDeploymentPackages") ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment

packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”, ”installDeploymentPackage”) ‹/blockquote› A holder of this permission can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=45,y=13) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework’s install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”listDeploymentPackages”) ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”, ”installDeploymentPackage”) ‹/blockquote› A holder of this permission can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=42,y=16) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework’s install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”listDeploymentPackages”) ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”,

”installDeploymentPackage”) ‹/blockquote› A holder of this permission can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=25,y=18) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework’s install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”listDeploymentPackages”) ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”, ”installDeploymentPackage”) ‹/blockquote› A holder of this permission can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=45,y=19) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework’s install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”listDeploymentPackages”) ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”, ”installDeploymentPackage”) ‹/blockquote› A holder of this permission

can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=39,y=22) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework’s install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”listDeploymentPackages”) ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”, ”installDeploymentPackage”) ‹/blockquote› A holder of this permission can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=25,y=23) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework’s install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”listDeploymentPackages”) ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”, ”installDeploymentPackage”) ‹/blockquote› A holder of this permission

can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=45,y=24) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework’s install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”listDeploymentPackages”) ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”, ”installDeploymentPackage”) ‹/blockquote› A holder of this permission can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

Unexpected tag, assume para (x=39,y=27) [p.167]

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework’s install/update/uninstall methods. ‹p› The permission uses a ‹filter› string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”listDeploymentPackages”) ‹/blockquote› A holder of this permission can access the inventory information of the deployment packages selected by the ‹filter› string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information. ‹blockquote› DeploymentAdminPermission(“‹filter›”, ”installDeploymentPackage”) ‹/blockquote› A holder of this permission

can install/upgrade deployment packages if the deployment package satisfies the ‹filter› string. ‹blockquote› DeploymentAdminPermission(“‹filter›”,”uninstall”) ‹/blockquote› A holder of this permission can uninstall deployment packages if the deployment package satisfies the ‹filter› string.

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
```

8.16.1 Summary

- DeploymentAdmin - This is the interface of Deployment Admin service. [p.166]
- DeploymentAdminPermission - DeploymentAdminPermission controls access to MEG management framework functions. [p.167]
- DeploymentPackage - The ResourcePackage object represents the a resource package (already installed or being currently processed) [p.168]
- ResourceProcessor - ResourceProcessor interface is implemented by processors handling resource files in deployment packages. [p.170]

8.16.2 public interface DeploymentAdmin

This is the interface of Deployment Admin service. *Service ID to register the service?*

8.16.2.1 public DeploymentPackage installDeploymentPackage(InputStream in)

in The input stream which where the deployment package can be read.

□ Installs a deployment package an input stream. If a version of that deployment package is already installed and the versions are different, the installed version is updated with this new version even if it is older. If the two versions are the same, then this method simply returns without any action.

Returns A DeploymentPackage object representing the newly installed/updated deployment package or null in case of error.

8.16.2.2 public DeploymentPackage[] listDeploymentPackages( )

□ Lists the deployment packages currently installed on the platform. MEGMgmtPermission(“”,”listDeploymentPackages”) is needed to access this method.

Returns Array of DeploymentPackage objects representing all the installed deployment packages.

8.16.2.3 public String location(String symbName, String version)

symbName The symbolic name of the bundle.

version The version of the bundle.

□ Gives back the location string generated from the bundle symbolic name and version. The deployment admin has to use this location string when installs an OSGi bundle with the given symbolic name and version.

Returns The location of the bundle generated from the paramaters.

8.16.3 public class DeploymentAdminPermission extends Permission

DeploymentAdminPermission controls access to MEG management framework functions. This permission controls only Deployment Admin-specific functions; framework-specific access is controlled by usual OSGi permissions (AdminPermission, etc.) In addition to DeploymentAdminPermission, the caller of Deployment Admin must in addition hold the appropriate AdminPermissions. For example, installing a deployment package requires DeploymentAdminPermission to access the installDeploymentPackage method and AdminPermission to access the framework's install/update/uninstall methods.

The permission uses a

string formatted similarly to the filter in RFC 73. The DeploymentAdminPermission filter does not use the id and location filters. The signer filter is matched against the signer of the deployment package, and the name filter is matched against the DeploymentPackage-Name header.

DeploymentAdminPermission(“

“,”listDeploymentPackages”) A holder of this permission can access the inventory information of the deployment packages selected by the string. The filter selects the deployment packages on which the holder of the permission can acquire detailed inventory information.

DeploymentAdminPermission(“

“,”installDeploymentPackage”) A holder of this permission can install/upgrade deployment packages if the deployment package satisfies the string.

DeploymentAdminPermission(“

“,”uninstall”) A holder of this permission can uninstall deployment packages if the deployment package satisfies the string.

8.16.3.1 public static final String ACTION_INSTALL_DP = “installDeploymentPackage”

8.16.3.2 public static final String ACTION_INVENTORY = “inventory”

8.16.3.3 public static final String ACTION_LIST_DPS = “listDeploymentPackages”

8.16.3.4 public static final String ACTION_UNINSTALL = “uninstall”

8.16.3.5 public DeploymentAdminPermission(String target, String action)

target Target string.

action Action string.

- Creates a new DeploymentAdminPermission for the given target and action.

8.16.3.6 public boolean equals(Object obj)

obj The reference object with which to compare.

- Checks two DeploymentAdminPermission objects for equality. Two permission objects are equal if their target and action strings are equal.

Returns true if the two objects are equal.

See Also `java.lang.Object.equals(java.lang.Object)`

8.16.3.7 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.

See Also `java.security.Permission.getActions()`

8.16.3.8 public int hashCode( )

- Returns hash code for this permission object.

Returns Hash code for this permission object.

See Also `java.lang.Object.hashCode()`

8.16.3.9 public boolean implies(Permission permission)

permission Permission to check.

- Checks if this DeploymentAdminPermission would imply the parameter permission.

Returns true if this DeploymentAdminPermission object implies the specified permission.

See Also `java.security.Permission.implies(java.security.Permission)`

8.16.3.10 public PermissionCollection newPermissionCollection( )

- Returns a new PermissionCollection object for storing DeploymentAdminPermission objects.

Returns The new PermissionCollection.

See Also `java.security.Permission.newPermissionCollection()`

8.16.4 public interface DeploymentPackage

The ResourcePackage object represents the a resource package (already installed or being currently processed)

8.16.4.1 public File getDataFile(Bundle bundle)

bundle A specified bundle

- Returns the private data area descriptor area of the specified bundle, which must be a part of the deployment package.

Returns The private data area descriptor of the bundle, or null in case of error.

8.16.4.2 public long getId( )

- Returns the identifier of the deployment package. Every installed deployment package has its own unique identifier. Once uninstalled, a deployment package will have an identifier value of -1.

Returns The ID of the resource package.

8.16.4.3 public String getName( )

- Returns the name of the deployment package.

Returns The name of the deployment package.

8.16.4.4 public String getVersion( )

- Returns the version of the deployment package.

8.16.4.5 public boolean isNew(Bundle b)

b Bundle to query.

- Allows querying whether the specified bundle is being newly installed by the current operation on the deployment package. This method gives resource processors the ability to query the effects of the current operation, which may be either commit or rollback. In the case of a rollback, the bundle would be returned to its previous version before the update operation was attempted.

Returns True if the bundle is newly installed by this deployment package, other False.

Throws `###WhatException` – Throws an exception if called outside an operation on this deployment package.

8.16.4.6 public boolean isPendingRemoval(Bundle b)

b Bundle to query.

- Allows querying whether this bundle is pending removal within the current operation on the deployment package. This method allows resource processors to query the effects of the current operation, which may be either commit or rollback. If the operation commits, the bundle will be uninstalled. In the case of a rollback, the bundle will not be uninstalled.

Returns True if the bundle is updated by this deployment package, otherwise False.

Throws `###WhatException` – Throws an exception if called outside an operation on this deployment package.

8.16.4.7 public boolean isUpdated(Bundle b)

b Bundle to query.

- Allows querying whether this bundle is being updated by the current operation on the deployment package. This method allows resource processors to query the effects of the current operation, which may be either commit or rollback. If the operation commits, the bundle will be uninstalled. In the case of a rollback, the bundle would be returned to its previous version before the update operation was attempted.

Returns True if the bundle is updated by this deployment package, otherwise False.

Throws `###WhatException` – Throws an exception if called outside an operation on this deployment package.

8.16.4.8 public Bundle[] listBundles( )

- Lists the bundles belonging to the deployment package. The returned list is not guaranteed to be up-to-date before the complete() method on ResourceProcessor is called.

Returns An array of bundles contained within the deployment package.

8.16.4.9 public void uninstall( )

- Uninstalls the deployment package. After uninstallation, the deployment package object becomes stale. This can be checked by using DeploymentPackage:getId(), which will return a -1 when stale.

8.16.5 public interface ResourceProcessor

ResourceProcessor interface is implemented by processors handling resource files in deployment packages. The ResourcePackage interfaces are exported as OSGi services. Bundles exporting ResourcePackage services may arrive in the deployment package or may be preregistered.

8.16.5.1 public static final int INSTALL = 1

The action value INSTALL indicates a desired install operation to be performed on a deployment package.

8.16.5.2 public static final int UNINSTALL = 3

The action value UNINSTALL indicates a desired uninstall operation to be performed on a deployment package.

8.16.5.3 public static final int UPDATE = 2

The action value UPDATE indicates a desired update operation to be performed on a deployment package.

8.16.5.4 public void begin(DeploymentPackage rp, int operation)

rp An object representing the deployment package being processed.

operation The operation in progress (INSTALL, UPDATE and UNINSTALL).

- Called when the Deployment Admin starts a new operation on the given deployment package, and the resource processor is associated with the package. Only one deployment package can be processed at a time.

8.16.5.5 public void complete(boolean commit)

commit True if the processing of the current deployment package was successful, and the changes must be made permanent. If False, the deployment package operation was unsuccessful, and the changes made during the processing of the deployment package should be removed.

- Called when the processing of the current deployment package is finished.

8.16.5.6 public void dropped(String name) throws Exception

name Name of the resource being dropped from the deployment package.

- Called when a resource, associated with a particular resource processor, had belonged to an earlier version of a deployment package but is not present in the current version of the deployment package. This provides an opportunity for the processor to cleanup any memory and persistent data being maintained for the particular resource.

Throws `Exception` – if the resource is not allowed to be dropped.

8.16.5.7 public void dropped( )

- Called when the resource processor is dropped entirely. This occurs when the processor is no longer associated with a deployment package, which provides an opportunity for the processor to cleanup any memory and persistent data being kept.

8.16.5.8 public void process(String name, InputStream stream) throws Exception

name The name of the resource relative to the deployment package root directory.

stream The stream for the resource. In case of uninstall it is null.

- Called when a resource is encountered in the deployment package for which this resource processor is registered.

Throws `Exception` – if the resource cannot be processed.

8.17 References

[15] *JAR File Specification*
http://java.sun.com/j2se/1.5.0/docs/guide/jar/jar.html

[16] *OSGi Framework*
####

Alternative API

I have made many remarks about the API that I consolidate here. Key objection is that the API is session based, but it is not modeled in objects. This means that the deployment package has session methods that are unusable in the normal case, not a good idea.

So I first propose to introduce a session object:

```
interface DeploymentSession {
   String getHeader(); // or on DP if persistent, next also
   String getResourceHeader(String path, String header );
   DeploymentPackage getSourcePackage();
   DeploymentPackage getTargetPackage();
   Directory getDirectory( Bundle bundle );
}
```

I added access to the headers because this is a very important communication path between the deployer and the resource processor. Evidence of this is that our "own" bundles "resource processor" has symbolic name and version, but we deny this capability to the RPs, bad.

The isPendingRemoval, etc. methods have been removed and replaced with access to the deployment package that will be replaced as well as the deployment package that replaces it. This makes it easy for a RP to figure things out. We do not have to spell this out in the API because very few RPs will need these methods (I expect).

The getDataFile (misnomer) is replaced with getDirectory for security reasons. This model allows the immplementation to do a doPrivileged when it creates the stream, random access file or reads the directory for subdirectories. The Directory interface looks like:

```
interface Directory {
   InputStream getInputStream(String file);
   OutputStream getOutputStream(String file);
   String [] getFiles();
   Directory [] getDirectories();
   Directory getParent();
   RandomAccessFile getRandomAccessFile(String file);
}
```

This mechanism is unfortunately necessary because java.io never considered security issues in its early design.

This makes the DeploymentPackage look like:

```
interface DeploymentPackage {
   long getId();
   String getName();
   String getVersion();
   void uninstall();
   String[] getResources();
   Bundle [] getBundles();
   ServiceReference getResourceProcessor(String resource);
   String getHeader(String name); // or on DA,
                                  // if not persistent
   String getResourceHeader(String path, String header);
   int hashCode();
   boolean equals(Object other);
}
```

Again, the isXXX(bundle) methods are gone. Added is a method to get all its resources. This is crucial information when something is wrong. One of the key frustration with Window's is that you have no idea what resources belong to an installed software package. This is also the reason to add a getHeader() method for the global section. This allows deployers to use copyright, name, help, documentation information (useing the Bundle-? headers?) in the manifest header. However, this will take memory ... so this is a tradeoff that must be made.

There is also a way now to get the RP that is associated with a resource. Again, this is crucial for fault finding and documentation (the ServiceRefrence return value is harmless).

With these objects, the RP would look like:

```
interface ResourceProcessor {
   void begin(DeploymentSession session) throws Exception;
```

```
    void install( DeploymentSession session,
      String resource, InputStream in) throws Exception;
    void uninstall( DeploymentSession session,
      String resource ) throws Exception;
    void complete(DeploymentSession session,boolean commit );
    void remove( DeploymentPackage pack );
    String [] getResources( DeploymentPackage pack );
}
```

Major change is the introduction of the session and use more common names for process/drop. Remove (dropped) is not inside a session. Also for recovery, the RP can list the resources it manages. Implementations must maintain this information anyway and it makes the UI a lot more pleasant.

Last, but not least, the Deployment Admin. I think the listDeploymentPackages should be called getDeploymentPackages, we normally use list when it takes a filter.

```
public interface DeploymentAdmin {
  DeploymentPackage installDeploymentPackage(InputStream
in);
  DeploymentPackage[] getDeploymentPackages();
}
```

The getDirectory call could be moved to DeploymentAdmin if it was useful in other contexts because the session is transient.

These are quite a lot of changes but none are fundamental. They will make the API however **much** easier to use imho.

####

9 Application Model Service Specification

Version 1.0

9.1 Introduction

The OSGi Application model is intended to simplify the management of an environment with many different types of applications that are simultaneously available. A diverse set of application models are a fact of life because backward compatibility and normal evolution require modern devices to be able to support novel as well as legacy applications. End users do not care if an application is an Applet, a Midlet, a bundle, a Symbian, or a BREW application. They are launching an application via an icon and expect the required functionality to be made available to them.

Therefore, an OSGi model is required to integrate different subsystems that provide execution environments for different application types. These environments are called *Application Containers*.

This specification defines the interaction between the application containers, a central administrative service and clients of this application Admin Service.

The OSGi Service Platform is an excellent platform on which to host different Application Containers. The powerful class loading and code sharing mechanisms available in the OSGi Service Platform can be used to implement powerful and extendable containers for Java based application models. Native code based application models like Symbian and BREW can be supported with proxies. The dynamic service registry provides an excellent mechanism that provides the interaction model between the Application Containers and the Application Admin.

9.1.1 Essentials

- *Generic Model* - The generic application model defines how all applications, regardless of type, are launched and stopped. This generic model allows a screen or desktop manager access to all executable content in a uniform manner.
- *Schedule* - A mechanism that allows the launching of applications at a pre-defined time, interval, or event.
- *Dynamic* - Detects installations and un-installations of applications in real time.
- *Locking* - Allows applications to be persistently locked so that they cannot be launched.

9.1.2 Entities

- *Application* - An Application object represents the actual running application for the Application Admin. For each Application Descriptor, many launched applications could be present.

Application Container - An Application Container service represents a specific application model. For example a Midlet Container contains the facilities to run Midlets. A Symbian container has access to the native environment of a device. Application Container services allow the installation of foreign formats as well as launching new applications. The implementations of this service should register an Application Descriptor object for each application that is available.

Application Handle - An Application Handle is used by either client code or container code. It is implemented by the Application Manager service. This public handle allows clients and container code to stop, suspend, and resume running applications. Each public Application Handle object has a 1:1 relationship to a private Application object that it controls. Containers must always use handles to control the state of the applications they manage.

Should you define Application Manager service?

Application Admin - The Application Admin service is the central access point for managing applications. This service can enumerate the running applications, launch new applications, schedule applications for a future launch, and provide a facility to lock an application so it can no longer be launched.

Application Descriptor - An Application Descriptor service provides access to the static (and localized) information of an application, such as the display name, icons, description, help and more. An Application Descriptor service must be registered by an Application Container service for each application that is resident.

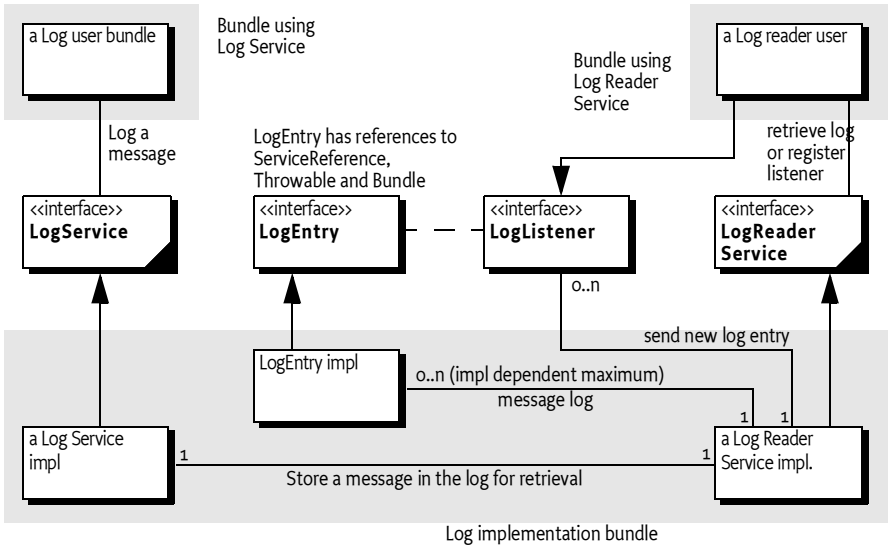
Should you define resident after this section? It's definition is buried in a subsequent section.

Scheduled Application - A Scheduled Application object is used to convey information about scheduled launchings and provides the ability to delete a schedule.

9.1.3 Entities

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

Figure 36 Log Service Class Diagram org.osgi.service.log package



9.1.4 Synopsis

The Application Admin uses the white board pattern and listens for Application Descriptor services. These descriptor services are registered by Application Container implementations. The list of Application Descriptor objects is made available to clients. When a client wants to launch an application, it asks the Application Admin to launch an Application Descriptor. The Application Admin first verifies that the requested application is unlocked.

If the application is unlocked, the Application Admin can perform pre-processing and security checks. If the launch is allowed, it creates an Application Handle object that represents the new instance. It then calls the right Application Container service to create an Application object, passing the Application Handle as a parameter. If this is successful, the application is started. The Application object is then registered with the OSGi service registry so its status becomes available to other bundles.

The client receives the Application Handle and can decide to pause, resume, or stop the application. The Application Handle also provides the current status of the application. If the application is stopped, the Application Handle becomes unusable and the Application object is removed from the service registry.

Applications that need to be launched at a specific event or time can register a Scheduled Application with the Application Admin.

9.2 Application Admin Service

The Application Admin service is the central access point for launching applications and obtaining information about resident applications. Resident applications are applications that can potentially be launched.

The previous statement is not as clear as the description of 'resident' that is provided below.

9.2.1 Resident Applications

This section appears to be more about Application Description service than Resident Applications.

The Application Admin provides an overview of resident applications with the getApplicationDescriptors() method. This method returns an array of the available Application Descriptor services. This set should be the same as the set of registered Application Descriptor services in the service registry. The Application Admin can filter this set based on security or implementation considerations. The Application Descriptor contains relevant information about the application.

The previous sentence seems superfluous, and seems like it would be better in the definition of Application Descriptor.

Each Application Descriptor has a unique identity which can be obtained with getUniqueID(). The unique ID is based on the same rules as a PID, see [1]. The Application Admin provides the getApplicationDescriptor(String) method to find a specific Application Descriptor service when the unique id is known.

The unique id should differ for different versions of the same application. If multiple Application Descriptors are registered with the same PID, the Application Admin must log a Framework Event WARNING. The Application Container should therefore ensure that the PID is actually unique.

The Application Descriptor also provides access to the name and version of the application. The (non-translated) name can be found with the getName() method. The version must follow the org.osgi.framework.Version semantics (major.minor.micro.qualifier).

The Application Descriptor signals the availability of an application, but it does not guarantee that the application can also be launched. There are certain situations in which the application has a number of prerequisites that are not yet met. For example, the application could have a thread running to initialize the database. In this case, the application is said to be resident. When the prerequisites are met, the application becomes available. A resident but not available application is disabled. Only when the application is available is it possible to launch it.

we need a way to signal this state change to the client. Following is proposal:

Few applications need access to this state change and the change is infrequent. Therefore, the client can listen to the service events of the Application Descriptor services. These applications therefore require ServicePermission[ApplicationDescriptor,GET]. The service property application.available is set to any value for available applications and must be absent for not yet available applications.

end proposal...

Application Descriptor services attributes, such as icons and descriptive texts, can be available in different locales. Access to this information is through a Map object. The getProperties(String locale) method returns a Map object that contains translated information. The parameter specifies a locale as used in java.util.Locale. Such locales consist of a triplet containing (language, country, variant). In the triplet, the more significant parts are specified first. The values for these locales are defined in ISO standards as referenced in the java.util.Locale class, for example:

en

en_US

du_BE

There are no mandatory keys in the properties map. However, a number of keys are standardized for common use. The keys are case sensitive and are listed below:

Is the 'properties map' the same as the map object mentioned above?

application.name [String] The user readable name of the application.

application.icon [String] URL to an icon representing the normal inactive but enabled state of an application. An icon may be a PNG, GIF or JPG file.

application.icon.running[URL] An icon for a running (launched) application application.icon.suspended[URL] An icon for a suspended application

application.icon.disabled[URL] An icon for an application that is resident but not currently available.

application.tooltip [String] Text that can be used as a tooltip or for another descriptive purpose.

application.help [URL] A URL to a help file.

application.vendor [String] Vendor description.

application.url [String] URL to an external web page that contains more information about the application.

application.copyright[String] Copyright string for this application

All the above icons can be available in different sizes. These icons use the above key names as the prefix and contain a suffix of 16, 32, 64, or 128. The suffix indicates the grid for which the icon is optimized. For example, an icon for an active application can be available as application.icon, application.icon.32, or application.icon.128. The name without a suffix should be optimized for 64x64 pixels. Screen managers are expected to scale the images appropriately.

Icons are returned as URL objects in the Map for security reasons. Clients of Application Admin are not likely to have permission to read the resources of any bundle, making it impossible for them to create URLs from strings that point to bundle resources. The same is true for a file area where a Midlet Container might store the Midlet icons.

The Application Descriptor can also specify its category. The category is a concise English language based string. The translation of categories needs to be provided by the environment. The value of this string is free. However, it is recommended to use the following values, which are case sensitive, to increase the success of grouping:

9.3 Category ValueApplication Description

communicationCommunication facilities

cryptographyPublic key, certificates

database Persistence

distributedDistributed computing such as CORBA, RMI, DCE

ecommerceElectronic shopping

games Entertainment, including proxies for games

graphics 2D/3D image viewer/manipulator

media Music/video player/editor

messagingMail, message queues

mobility Functions such as positioning

network Implementations of network protocols

office Office applications

payment Electronic payments

system System monitoring/manipulating tools

test Testing

tutorial Informational and educational

utility Support applications providing a utility

The complete list of OSGi categories can be found at http://member-cvs.osgi.org/docs/reference.html#category

9.4 Launching an Application

Applications are launched with the help of an Application Descriptor service. The launch method is, however, not on the Application Descriptor service but on the Application Admin service. This is because of the following:

Resource constraintsThe OSGi Service Platform is designed to run on constrained devices. Embedded devices are quite different from desktops or servers where memory is not a practical constraint. In a constrained environment, it might be necessary to shutdown some applications or prepare the environment in a certain way before an application is launched. The Application Descriptor service is implemented by the Application Container. A launch method on the Application Descriptor would thus bypass the Application Admin service.

Security The Application Admin may implement a security scheme that prohibits users or devices from launching applications.

The method to launch an application is launchApplication(ApplicationDescriptor,Map). The Application Admin performs the following actions:

This should be an ordered list. An ordered list implies sequential execution.

Prepare If the application is locked, returns with null immediately. Otherwise, performs any preparations deemed necessary.

Application HandleCreates an Application Handle. This is the object used to identify the application from the client to the Application Admin and this object can be used to control the launched application.

Find container Finds the appropriate container that implemented the Application Descriptor. This mapping is achieved with the application.type property that is registered both with the Application Descriptor service as well as the Application Container service.

Create Creates a new application in the container with the createApplication(ApplicationDescriptor,Map,ApplicationHandle) method. If this creation fails with an exception, the exception must be rethrown and the application is not launched. The SingletonException is a specific exception used to signal that there is already such an application running and that the application is a singleton (i.e. only one running instance allowed). The Application Container can find the specific Application Descriptor from the Application Handle objects method getApplicationDescriptor(). The Application Admin will then register the Application Handle in the service registry. The create operation returns an Application object.

Start Start the application by calling the startApplication() method on the returned Application object.

What is our thread model. Do we call the Application startApplication on the same thread or does it get its own thread?

Return Returns the Application Handle object associated with the application.

This sequence is depicted in Figure 1.

Figure 1 Sequence diagram for launching an application

The previous picture shows how an application is launched. Once it is launched, the application is brought in the RUNNING state. The startApplication() method on the Application object has been called before it returns to the client. startApplication() must be called in the same thread as the caller with the full security context of the caller. For example, the Application Admin service must not perform a doPrivileged before calling the createApplication method.

The Application Container can start the real application in another thread or process.

Directly after the startApplication() method has successfully returned, the Application Admin must register the Application Handle with the service registry. It must provide the following properties with this registration:

application.pid [String] The unique identity of the associated Application Descriptor

application.state [String] The state of the application, which is either RUNNING or SUSPENDED.

If the state of the handle changes due to a suspend or resume action, the Application Admin services must update the application.state property. This property only shows the RUNNING and the SUSPENDED state. The transient states are not reflected in the property. The final state should cause the Application Handle service to be unregistered.

9.5 Application States

OSGi Service Platforms are targeted at small and constrained environments. These types of environments require careful usage of resources and CPU load. At certain times, for example, during an incoming call on a mobile, it is necessary to ask all running applications to minimize their resource consumption and CPU usage. The state in which the applications are minimized is called suspended. The Application Handle has methods to move the application in and out of the suspended state.

The Application Handle is returned to the client in the RUNNING state. The client may stop the application at any time, while also in a suspended state. Once an application is stopped, it can no longer change state. The resulting state diagram is depicted in Figure 2 and is explained in the following section.

Figure 2 Application state diagram

RUNNING The application is active, and the startApplication() method on the Application object has been called. Calling the stopApplication on the Application Handle object moves the state to STOPPING and then the stopApplication() method is called on the Application object. The Application Handle object's resumeApplication() is ignored in this state.

SUSPENDING The client has called suspendApplication() on the Application Handle object. The Application Manager is in the process of calling suspendApplication() on the Application object. The Application Handle must ignore any further state change requests while in this state . After the suspendApplication() of the Application object returns, the state is automatically set to SUSPENDED.

SUSPENDED The application is suspended. Further suspendApplication() calls on the Application Handle object are ignored . A resumeApplication() call causes the state to be moved to RESUMING and the resumeApplication() call on the Application object will be called. After this call returns, the state is moved to RUNNING again. The Application Handle object must ignore any suspendApplication() calls in this mode. If the stopApplication() is called in this state, then the state is moved to STOPPING and the shutdown of the application must progress.

RESUMING The Application object is running the resumeApplication() method. After the return of this method, the state is automatically moved to RUNNING.

STOPPING If this state is entered, the Application Handle object is unregistered from the service registry. The Application object's stopApplication() is then called. If it returns, the Application Handle moves to the final state. The final state must cause all methods to throw an Illegal State Exception.

What about the case above where the Application object's stopApplication() does not return?

State transitions must take place regardless of whether the Application object's life cycle method throws an exception or not. Any such exceptions should be logged because the Application should not throw exceptions during these calls.

If the Application Container's model does not have a state similar to SUSPENDED and RESUMED, then the suspendApplication() and resumeApplication() calls may be ignored by the Application Container.

9.5.1 Locking

Applications can be locked. Locking in this context means that an application must not be launched when requested. The following methods are involved in the locking process:

lock(ApplicationDescriptor)Marks the application to be locked. This mark is persistent and associated with the PID of the Application Descriptor. If the application was already locked, this is a no-op. After this call, the launch of the application returns null. The isLocked() method returns true for any Application Descriptor with the same PID. The lock method must only control future launches. Running applications are unaffected.

unlock(ApplicationDescriptor)

Unlocks the application. After this call the associated application can be launched again and isLocked() returns true for this Application Descriptor.

isLocked(ApplicationDescriptor)

Answers (do you mean 'returns'?) the locked state of the Application Descriptor.

9.6 Access to Launched applications

Clients receive an Application Handle object when they have successfully launched an application. This handle provides access to the state and can be used to change the state.

Other clients that want to find out, or follow, the state of running applications should use the service registry. The Application Admin service registers the Application Handle in the OSGi service registry and modifies the properties if the state moves from RUNNING to SUSPENDED or vice versa. If the application is stopped, the registration is removed. The service registry event mechanism thus provides a convenient way to track active applications.

Scheduling Applications

The Application Admin service provides a convenient method to schedule the launch of an application at a specified time. The following methods are available to manage the schedules.

addScheduledApplication(ApplicationDescriptor,Map,Date)

Add a new schedule to the current list. The schedule specifies that the Application Descriptor should be used to launch a new application at the given time. The schedule is recorded persistently.

getScheduledApplications()Returns a list of Scheduled Application objects that have not yet expired. These objects can be used to inspect and remove schedules.

When an application is scheduled, the Application Admin will launch the application with the given parameters at the specified time.

what about the security context? I think it would be handy to define the AccessControlContext in which this is going to play, however, the schedule is persistent so we would need some user identifier?

9.7 Application Container services

The Application Container service can define different application models, including native, Midlets, OSGi Mobilets, BREW etc. Each container must register an Application Container service with the following properties:

service.pid [String] A unique identity for this container

application.type [String] The type of applications that the container can handle. This string must be identical to the application.type property that is registered with an Application Descriptor service from this Application Container.

Once the Application Container service is registered, it must register Application Descriptor services for each application it can launch. This way, the Application Container service informs the Application Admin of available applications. For example, if a Midlet Application Container registers an Application Descriptor service for each Midlet that can be launched. The registration of an Application Descriptor must set the following properties:

service.pid [String] The PID of the service, which must be equal to the unique id of the Application Descriptor.

application.type [String] The identifier of the container type, which must be identical to the named registration property of the associated Application Container service.

The Application Descriptor implementations are specific for each container. This flexibility allows to adapt closely to the existing application model.

I'm not sure what the previous sentence is trying to convey. Do you mean: The Application Descriptor implementations are specific for each container. This flexibility allows you to adapt your implementation to closely resemble the existing application model.???

When an application needs to be instantiated, the Application Admin calls createApplication(ApplicationHandle,Map). The Application Container then gets the Application Descriptor from the given Application Handle and creates an instance of the given application. However, this application must not be started yet. The Application Admin will start the application later, after successful creation. If the application can accept parameters, then the given Map object is converted to the format the application can understand. The container must then return an Application object. An Application object remains a token between the Application Admin and the container and it is not passed to clients. The container can therefore implement the Application interface directly on its own control object, or, in special cases, on a base class of the real application code.

A suspended application must minimize its resource consumption. Any threads it holds should be suspended and the Application Container should minimize resource consumption. If the application is stopped, the Application Container must clean up all resources the application has used.

Are there any outside things that can happen that we can tell the container? I.e all its threads are suspended (though that likely creates deadlocks)???

The Application Container is fully responsible for the state of its applications. The Application Admin does not handle the case where a stopApplication(), resumeApplication(), or suspendApplication() fails. It is the responsibility of the Application Container to handle possible errors from the application, resolve those problems, and log these errors accordingly. The Application Admin must serialize the calls to the Application object as well as remove any repeats. For example, the Application Container can assume that none of the Application methods is called simultaneously on different threads, nor does it have to check that an application is already in the requested state.

9.8 Installing and Uninstalling

installation, we need to discuss with deployment how to handle the foreign installation model. Or do we stick to the calls we have now?

Security

Service Permission

The ServicePermission[ApplicationAdmin,REGISTER] must be restricted to the Application Admin bundle. ServicePermission[ApplicationAdmin,GET] can be given to all bundles. ServicePermission[ApplicationContainer,GET] must be reserved for the Admin Service implementation bundle. ServicePermission[ApplicationContainer,REGISTER] should be provided only to trusted bundles.

doPrivileged

Do you need to define doPrivileged?

The Application Admin service must not perform a doPrivileged before it launches an application. An application must be started in the full context of the caller. A caller should be allowed to run the launch operation in a context that depends on a user. For example, the client must be able to perform a doPrivileged() call that restricts the context of the launched application.

This rule implies that both Application Containers as well as the Application Admin need to have AllPermission or at least be more powerful than they need to be for their own operations. Any permission for these bundles is a restriction on all applications that will be started.

The Application Admin service can then provide this information in a unified way to client applications that need to enumerate and launch applications, regardless of whether these applications are Midlets or native Symbian. An example of this type of application is a screen or desktop manager.

9.9 The Application Admin Service Interface

9.10 Security

.

9.11 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; version=1.0
```

9.11.1 Summary

- Application - An application in the MEG environment. [p.187]
- ApplicationContainer - [p.187]
- ApplicationContext - [p.188]
- ApplicationDescriptor - Descriptor of an application. [p.188]
- ApplicationHandle - [p.188]
- ApplicationManager - [p.189]
- ApplicationManagerPermission - This class implements permissions for the Application Manager [p.191]
- MEGApplication - A MEG Application, a.k.a Meglet [p.191]
- MEGApplicationContext - [p.192]
- ScheduledApplication - A scheduled application contains the information of future registered applications. [p.192]
- SingletonException - [p.192]

9.11.2 public interface Application

An application in the MEG environment.

9.11.2.1 public void resumeApplication( ) throws Exception

9.11.2.2 public void startApplication( ) throws Exception

9.11.2.3 public void stopApplication( ) throws Exception

9.11.2.4 public void suspendApplication( ) throws Exception

9.11.3 public interface ApplicationContainer

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

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

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

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

9.11.4 public interface ApplicationContext

9.11.4.1 **public Map getLaunchArgs( )**

9.11.5 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.

9.11.5.1 **public String getCategory( )**

- Return the category of the application.

  The following list of application types are predefined:

  - APPTYPE_GAMES
  - APPTYPE_MESSAGING
  - APPTYPE_OFFICETOOLS

9.11.5.2 **public String getContainerID( )**

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

  - MEG (?)
  - Midlet
  - Doja

9.11.5.3 **public Map getContainerProperties(String locale)**

9.11.5.4 **public String getName( )**

- Returns the display name of application

9.11.5.5 **public Map getProperties(String locale)**

9.11.5.6 **public String getUniqueID( )**

- Get the unique identifier of the descriptor.

9.11.5.7 **public String getVersion( )**

- Returns the version descriptor of the service.

9.11.6 public interface ApplicationHandle

9.11.6.1 **public static final int NONEXISTENT = 5**

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

9.11.6.2 public static final int RESUMING = 3

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

9.11.6.3 public static final int RUNNING = 0

The application instance is running

9.11.6.4 public static final int STOPPING = 4

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

9.11.6.5 public static final int SUSPENDED = 2

The application instance has been suspended

9.11.6.6 public static final int SUSPENDING = 1

The application instance is being suspended

9.11.6.7 public void destroyApplication( ) throws Exception

9.11.6.8 public ApplicationDescriptor getAppDescriptor( )

9.11.6.9 public int getAppStatus( )

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

9.11.6.10 public void resumeApplication( ) throws Exception

9.11.6.11 public void suspendApplication( ) throws Exception

9.11.7 public interface ApplicationManager

9.11.7.1 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

9.11.7.2 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

9.11.7.3 public ApplicationDescriptor[] getAppDescriptors( )

- Get the descriptors of all the installed applications.

9.11.7.4 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

9.11.7.5 public String[] getSupportedMimeTypes( )

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

Returns A list of MIME types supported in the system

9.11.7.6 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

9.11.7.7 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

9.11.7.8 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

9.11.7.9 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

9.11.8 public class ApplicationManagerPermission extends BasicPermission

This class implements permissions for the Application Manager

9.11.8.1 public static final String CONTENT = “content”

9.11.8.2 public static final String ENUMERATE = “enumerate”

9.11.8.3 public static final String LAUNCH = “launch”

9.11.8.4 public static final String PROVIDE = “provide”

9.11.8.5 public static final String SCHEDULE = “schedule”

9.11.8.6 public ApplicationManagerPermission(String actions)

actions - read and write

- Constructs a ApplicationManagerPermission.

9.11.8.7 public ApplicationManagerPermission(String name, String actions)

name - name of the permission

actions - read and write

- Constructs a ApplicationManagerPermission.

9.11.9 public abstract class MEGApplication implements Application , EventHandler

A MEG Application, a.k.a Meglet

9.11.9.1 **public MEGApplication(MEGApplicationContext context)**

9.11.9.2 **public abstract void handleEvent(Event event)**

9.11.9.3 **public abstract void resumeApplication( ) throws Exception**

9.11.9.4 **public abstract void startApplication( ) throws Exception**

9.11.9.5 **public abstract void stopApplication( ) throws Exception**

9.11.9.6 **public abstract void suspendApplication( ) throws Exception**

9.11.10 public interface MEGApplicationContext extends ApplicationContext

9.11.10.1 **public ApplicationManager getApplicationManager( )**

9.11.10.2 **public EventAdmin getEventAdmin( )**

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

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

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

9.11.11 public interface ScheduledApplication

A scheduled application contains the information of future registered applications.

9.11.11.1 **public ApplicationDescriptor getApplicationDescriptor( )**

9.11.11.2 **public Date getDate( )**

9.11.11.3 **public void remove( )**

9.11.12 public class SingletonException extends Exception

9.11.12.1 **public SingletonException( )**

9.12 References

[1] Configuration Admin Service, OSGi Service Platform, R4

[2] Application Model Specification

10 Meglets Specification

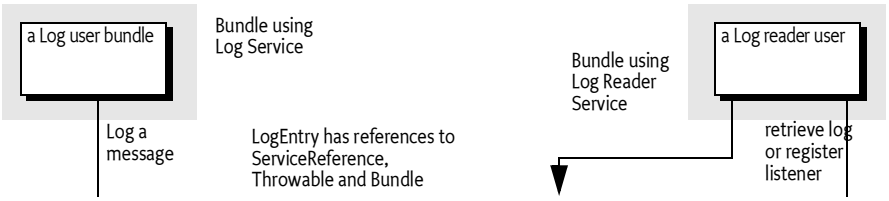
Version 1.0

10.1 Introduction

10.1.1 Entities

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

Figure 37 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; version=1.0
```

10.4.1 Summary

- Mobilet - A MEG Application, also called a Meglet. [p.193]
- MobiletFactory - [p.194]

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

multi threading not allowed, no detection

11.1 Introduction

The OSGi Service Platform is a generic, service-centric execution environment. From its very inception the OSGi Service Platform had an architecture for remote management in place. This architecture is based on a Management Agent contained in, and deployed as, a Management Bundle. The management of an OSGi-based environment is then performed by a Remote Manager, communicating with the Management Agent over a mutually agreed protocol.

This capability is pivotal for an environment as dynamic and as flexible as an OSGi Service Platform, allowing provisioning of new services and replacement of existing ones on the fly with protocols selected to be optimal for the occasion. This flexible model is crucial for an execution environment that is suitable for a large number industries that all have their own *industry standard protocols*.

With the OSGi Service Platform being adopted on a larger and larger scale, in more and more industries, the complexity of the overall systems increase. Same server products can be expected to be deployed by multiple operators, and be required to work with devices from several manufacturers. The software on the devices can be sourced not only from manufacturers and operators, but increasingly from third-party developers. This software can go far beyond simple games, and can even constitute a part of the device's own infrastructure.

To manage this increased complexity. the Management Agent is likely to require uniform access to the device management information over a simple and consistent API. This API must allow components to extent the management model.

This document introduces such an API, as well as an infrastructure of plugins necessary to make device management highly dynamic and extensible.

11.1.1 Entities

- *Device Management Tree* – The Device Management Tree (DMT) is the logical view of manageable aspects of an OSGi Environment, structured in a tree with named nodes.
- *Dmt Admin* – A service through which the DMT can be manipulated. It is used by local managers or by protocol adapters that forward DMT opera-

tions. The Dmt Admin service forwards selected DMT operations to Data Plugin and execute operations to Exec Plugin services.

- *Alert Sender* – This service can be used by the Dmt Data Plugin services, Dmt Exec Plugin services, and clients to send a notification to a remote manager.
- *Plugin* – Services which take the responsibility over a given subtree of the DMT. There are Data Plugin services and Exec Plugin services.

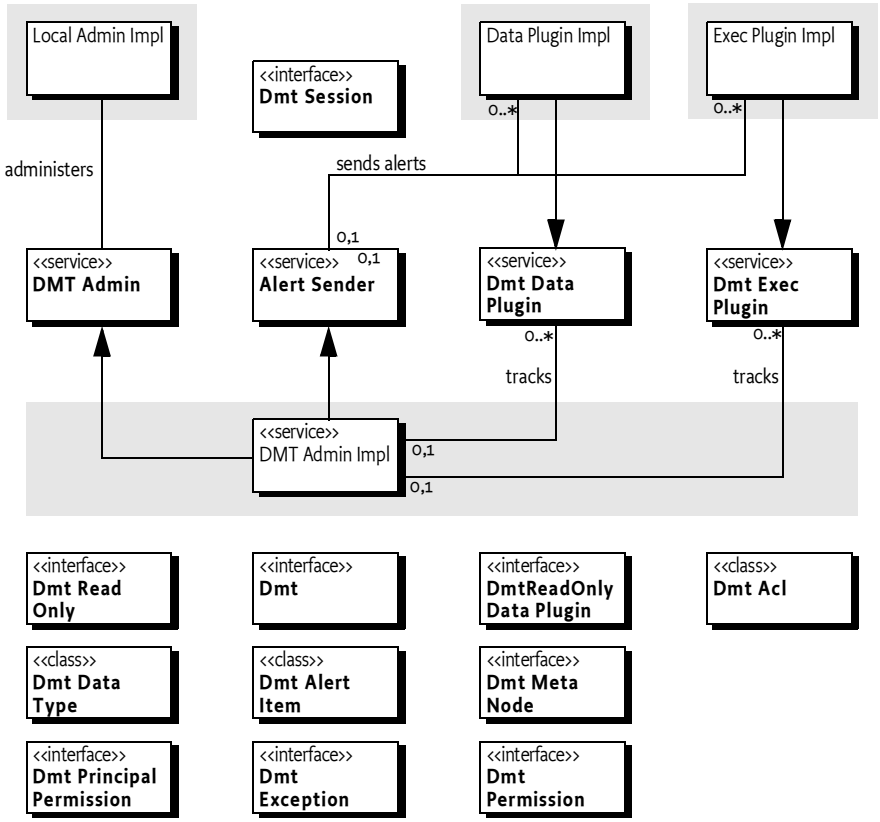
Can a data and exec plugin overlap?

- *Data Plugin* – A Plugin that can handle data operations.
- *Exec Plugin* – A Plugin that can handle exec operations.
- *Session* – A session is grouping a set of operations with optional transactionality and locking. Dmt Session objects are created by the Dmt Admin service and forwarded to the plugins for each operation.
- *Principal* – The principal represents the identity of the optional initiator of a Dmt Session. Principal's are necessary for security reasons.
- *ACL* – An Access Control List (ACL) is a set of principal to permitted operations associations.
- *Meta Node* – Information provided by plugins about a node to perform validation and to provide assistence to users when these values are editied.
- *Back-end* – The DMT is only a view of the management state of the device. The actual data is available either from a plugin, stored in a data back-end service, or stored in the Dmt Admin service's private storage area. An example for a data back-end service is the Configuration Admin service: the data in the configuration data store is made available for reading and writing through the DMT.

What is the difference with a Data Plugin? Is this term required ...

- *Local Manager* – A bundle which uses the Dmt Admin service directly to read or manipulate the DMT.
- *Protocol Adapter* – A bundle that communicates with a management server external to the device and uses the Dmt Admin service to operate on the DMT.

Figure 38 DMT Admin Service Class org.osgi.service.dmt package



Picture still needs work

Are there any mandatory nodes in the tree?

11.2 The Device Management Model

The most important decision in determining any fundamentally new architecture is choosing a single meta-data model. A model that must express a common conceptual and semantic view for all consumers of that architecture. In the case of networked systems management there are a number of meta-data models to choose from:

- *SNMP* – The best-established and most ubiquitous model for network management. See [22] *SNMP* for more information.
- JMX, a generic systems management model for Java, a de-facto standard in J2EE management. See [19] *Java™ Management Extensions Instrumentation and Agent Specification* for more information.

- *JSR 9 FMA* – Federated Management Architecture (FMA) [5], another Java standard originating in storage management. See [20] *JSR 9 - Federated Management Architecture (FMA) Specification*
- *CIM/WBEM* – Common Information Model (CIM) and Web-Based Enterprise Management, a rich and extensible management meta-model developed by the DMTF. See [21] *WBEM Profile Template, DSP1000.*

For various reasons none of these models enjoy any significant mind share within the mobile device community. Some, like SNMP, are primitive and very limited in functionality. Some, such as JMX and FMA, are too Java-centric and not well-suited for mobile devices.

One model that appears to have gained an almost universal acceptance is the Device Management Tree (DMT), introduced in support of the OMA DM protocol (formerly known as SyncML DM).

OMA DM provides a hierarchical model, like SNMP, but it is more sophisticated in the kinds of operations and data structures it can support. It is also designed with the restrictions and specifics of small devices in mind.

Using XML?????? How honest do we have to be? :-)

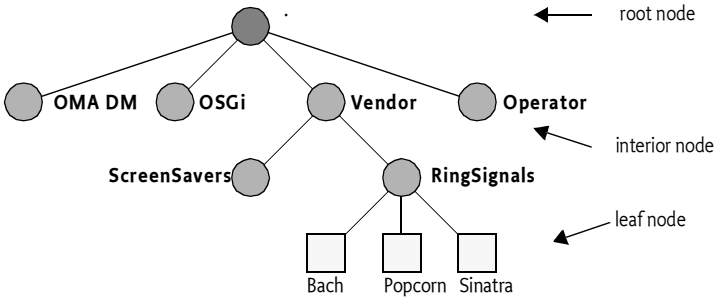
11.2.1 The Device Management Tree

The standard-based features of the DMT model are:

- The Device Management Tree consists of interior nodes and leaf nodes. Interior nodes have children and leaf nodes have values.
- All nodes have a set of properties: Name, Title, ACL, version, size, and ###
- The storage of the nodes is undefined, nodes typically map to registers, settings, etc.
- A node's name must be unique within its parent.
- Meta nodes can be defined as children of a particular node, with no specific names defined, i.e. they can match any new node.
- Only leaf nodes have values
- Base value types (called *formats* in the standard) are
  - 32 bit Integer
  - Unicode string
  - Boolean
  - Binary data
  - XML fragments
- Leaf nodes in the tree can have default values
- Meta nodes have allowed access operations defined (`Get`, `Add`, `Replace`, `Delete` and `Exec`)
- Nodes can have Access Control Lists (ACLs), associating operations allowed on those node for a particular principal.

This list is a bit unstructured

Figure 39 *Device Management Tree example*



Based on its industry acceptance and technical features, the DMT model was chosen by the OSGi Alliance as the uniform meta-data and operational model. In this capacity it is considered separately and independently from OMA DM or any other provisioning protocol, and underlies all local and remote device management operations on the OSGi Environment.

Users of this specification should be familiar with the concept of the Device Management Tree and its properties and operations as defined by OMA DM, see [17] *SyncML Device Management Tree and Description, Version 1.1.2*.

I am not sure how we split the responsibility. We inherit the semantics of the OMA DM but I also think we should be complete. What happens when there is a deviation?

11.2.2 Extensions

The Device Management Tree model as described above does not allow for enforcement of the data integrity. Therefore, this specification introduces additional attributes in the meta nodes, refining semantics of both interior and leaf nodes. The following constraints have been added to the meta data:

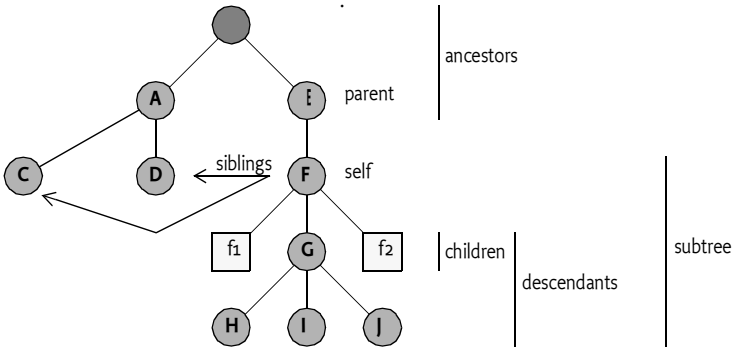
- *Range* – Max/min. values for integers
- *Length* – Max length for string values and binary values
- *Enumeration* – Valid values lists
- *Patterns* – Regular expressions to be matched for string values
- *Referential Integrity* – Referential integrity constraints provide relations between values and other nodes.

Length, enumerations and patterns are possible for both node values as well as node names.

11.2.3 Tree Terminology

In the following sections, the DMT is discussed in many places. It is therefore important to have well defined terms. The different terms are shown in Figure 40.

Figure 40 *DMT naming, relative to node F*



The terms are defined as follows:

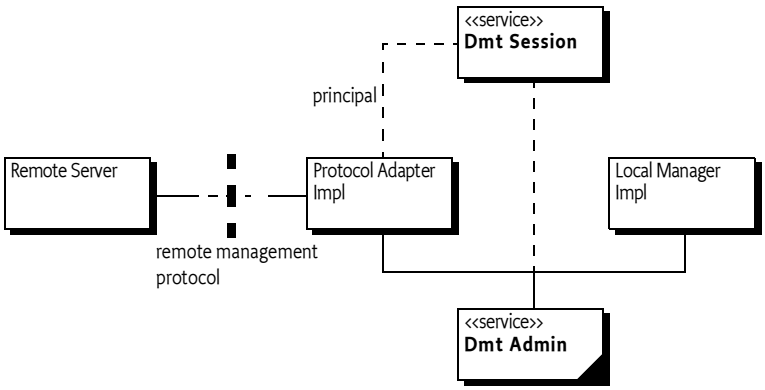
- *ancestors* – All nodes that are above the given node ordered in proximity. I.e. the closes node is first in the list. In the example, these list is [./E, .]
- *parent* – The first ancestor, in the example this is ./E.
- *children* – A list of nodes that are directly beneath the given node without ordering. For node F this list is { ./E/F/f1, ./E/F/f2, ./E/F/G }.
- *siblings* – An unordered list of nodes that have the same parent. All siblings must have different names. For F, this is { ./A/D,./A/C }
- *descendants* – A list of all nodes below the given node. For F this is { ./E/F/f1, ./E/F/G, ./E/F/f2, ./E/F/G/H, ./E/F/G/I, ./E/F/G/J }
- *subtree* – The given node plus the list of all descendants. For F this is { F, ./E/F/f1, ./E/F/G, ./E/F/f2, ./E/F/G/H, ./E/F/G/I, ./E/F/G/J }
- *overlap* – Two given URIs overlap if they share any nodes in their subtrees. In the example, the subtree F and C overlap.

11.2.4 Actors

There are two typical users of the Dmt Admin service:

- *Remote manager* – The typical client of the Dmt Admin service is a *Protocol Adaptor*. A management server external to the device can issue DMT operations over some management protocol. The protocol to be used is not specified by this specification. However, the likely ones used in practice will be OMA DM and IOTA. The operations reach the service platform through the protocol adaptor, which forwards the calls to the Dmt Admin service.
- *Local Manager* – A bundle which uses the Dmt Admin service to operate on the DMT, for example a GUI application that allows the end user to change settings through the DMT.
  Though possible, it may be easier to directly use the services that underly the DMT. Many of the management features available through the DMT are also available as services. These services abstract the callers from the underlying details of the DMT structure. As an example, it is more straightforward to use the Monitor Admin service than to operate upon the monitoring subtree.
  The local management application might listen to Dmt Events if it is interested in updates in the tree made by other entities.

Figure 41 Actors



11.3 The DMT Admin Service

This specification is based on the concept of a Dmt Admin service for operating on the DMT of an OSGi based device. The Dmt Admin API is closely modelled after the OMA DM protocol: the operations for Get, Replace, Add, Delete and Exec are directly available.

Access to the DMT is session based to allow for transactionality. The sessions are in principle concurrent and can be transactional. The client indicates to the Dmt Admin service what kind of session is requested: Shared, exclusive or atomic.

- *Exclusive Update Session*– Two or more updating sessions can not update the same part of the tree simultaneously. An updating session must acquire an exclusive lock on the subtree which blocks the creation of other sessions that want to operate on an overlapping subtree.

How does that work with referential integrity constraints?

- *Multiple Readers Session* – There can be any number of concurrent read only sessions, but the ongoing read only sessions will block the creation of an updating session within the same subtree.

Do we have to specify anti-starvation rules? I.e. ordering?

what about priorty inversion?

- *Optional Transactions* – This specification does not mandate two phase commit transactionality. The lack of real transaction support can lead to error situations which are described later in this document, see *Plugins and Transactions* on page 216. ### wrong link

Although the DMT represents a persistent data store with transactional access and without size limitations, the notion of the DMT should not be confused with a general purpose database.

The intended purpose of the DMT is to provide a *dynamic view* of the management state of the device, this is what the DMT model and the Dmt Admin service are designed for. Other kinds of usage, like storing and sharing generic application specific data, is strictly discouraged.

11.4 Manipulating the DMT

11.4.1 The DMT Addressing URI

Nodes in the DMT are addressed using a URIs. URIs are defined in [23] *RFC 2396 Uniform Resource Identifiers (URI): Generic Syntax*. This RFC defines the following set of acceptable:

```
uri-set = [-a-zA-Z0-9_.!~*'()]
```

All other characters must be escaped. For characters of the US ASCII codeset (0x00 to 0x7F) this is straightforward: The character is replaced with a percent sign ('%' \u0025) and a 2 hex digits. E.g. the space character (which is not in the uri-set, can be represented as %20. For characters outside the US ASCII range, it is necessary to specify a character set because RFC 2396 does not define this character set. For this specification, the character set is defined to be UTF-8.

For example, to insert a copyright symbol (© \u00A9), it is necessary to escape the copyright symbol.

```
./ACME ©/A/x      ./ACME %C2A9/A/x
```

Yes?

Nodes are addressed by presenting a *relative URI* for the requested node. Relative URIs do not specify the scheme and its parameters. The base URI is assumed to point to the root of the management tree in the device.

The URI for a node is constructed by starting at the device root and, as the tree is traversed down to the addressed node. Each node name is appended to the previous ones using a slash (/ \u002F) as the separating character.

For example, to access the Bach leaf node in the RingTones interior node from Figure 39 on page 199, the URI must be:

```
./Vendor/RingSignals/Bach
```

The URI must be given with the root of the management tree as the starting point. URIs used in the DMT must be treated and interpreted as *case sensitive*. I.e. ./Vendor and ./vendor are two different nodes. The following restrictions, taken over from OMA DM, on URI syntax are intended to simplify the parsing of URIs.

- *No End Slash* – A URI must not end with the delimiter slash (/ \u002F). This implies that the root node must be denoted as . and not ./.
- *Parents* – A URI must not be constructed using the character sequence ../ to traverse the tree upwards.
- *Single Root* – The character sequence ./ must not be used anywhere else but in the beginning of a URI.

11.4.2 Locking and Sessions

The Dmt Admin service is the main entry point into the Device Management API. Its sole usage is to create sessions.

The simple case to get a session is to get a session on a specific subtree. Such a session can be created with the getSession(String) method. This method creates an updating session with an exclusive lock on the given subtree.

The URI parameter is the subtree address. If null, it addresses the root of the DMT. In principle all nodes can be reached from the root, so specifying an interior node is not necessary. However, Dmt Admin implementation can use this information to optimize concurrent access.

What happens if the URI addresses a leaf node?

The getSession methods can block depending on the presence of other opened sessions with overlapping subtrees. Therefore, if the default locking mode of a session (exclusive) is too restrictive, it is possible to specify the locking mode with the getSession(String,int) method. This method supports the following locking modes:

- LOCK_TYPE_SHARED – Creates a *shared session*. It is limited to read-only access to the subtree which means that multiple such sessions are allowed to read the given subtree at the same time.
- LOCK_TYPE_EXCLUSIVE – Creates an *exclusive session*. The lock guarantees full read write access to the tree. However, such sessions can not share the tree with any other session.
- LOCK_TYPE_ATOMIC – This creates an *atomic session* with an exclusive lock on the subtree but with added transactional functionality. Operations on such a session must either succeed together or fail together.

Is this optional or mandatory? Will this fail if the system does not suppor real transactions?

The Dmt Admin service must lock the subtree. If the requested subtree is not accessible, the getSession method must block until the subtree becomes available. The method does not throw an exception due to a subtree being locked, although it may time out in an implementation specific manner.

Why is this not specified? Looks rather important to me.to know what happens then?

As a simplification, the Dmt Admin service can lock the entire tree irrespective of the given subtree. However, for performance reasons, implementations should provide more fine-grained locking.

Plugins must be exclusively opened and associated to a single session. Once a plugin is associated with a session, no other session must open that plugin. Otherwise the plugin would be unable to determine on receiving a rollback or close call which session it relates to. Locking a subtree does not prevent this situation because a plugin could handle multiple interior nodes that address non-overlapping subtrees.

Is this also true for read only? yes

should go the Plugin section

Persisting the changes of a session works differently for exclusive and atomic sessions. Changes to the subtree in an atomic session are not persisted until the close method of the session is called. Also, once an error is encountered, all successful changes must be rolled back automatically.

Changes in an exclusive session are persisted immediately after each operation. Errors do not automatically rollback any changes made in a session.

Due to locking ant transactional behavior, it is required to close a session once it is no longer used. This must be done for all session types. Locks must always be released even if the close method throws an exception.

Alternatively, the rollback method can be called. For non-atomic sessions this is identical to the close method.

Once a session is closed, after a close or rollback method is called, no further operations are allowed and all methods must throw an Illegal State Exception when called.

The close method can be expected to fail even if all or some of the individual operations were successful. This can happen due to multi-node 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 the close method is executed, the close method can fail and throw an exception. The details of how an implementation specifies such constraints is outside the scope of this specification. In these cases an atomic session must rollback all the changes made in that session.

Should the previous paragraph be here? Better place?

11.4.3 Associating a Principal

Protocol adapters must use the getSession(String,String,int) method which features the principal as the first parameter. The principal identifies the external entity on whom's behalf the session is created. This server identification string is determined during the authentication process in a management protocol specific way.

For example, the identity of the OMA DM server can be established during the handshake between the OMA DM agent and the server. In the simpler case of OMA CP protocol, which is a one-way protocol based on WAP Push, the identity of the agent can be fixed a value.

The caller of this constructor must have the Dmt Principal Permission for the given principal. This Dmt Permission is used to enforce that only trusted entities can act on behalf of remote managers.

The other two forms of the getSession method are meant for local management applications where no principal string can be used. No special permission is defined to restrict the usage of these methods. However the callers that want to execute device management commands need to have the appropriate Dmt Permissions. See *Operational Permissions* on page 226 for more information.

not clear, do you need DmtPermission to open or not?

11.4.4 Relative Addressing

All tree operation methods are found on the session object. Most of these methods accept a node relative URI as their first parameter, for example the method isLeafNode(String nodeUri).

This URI is always relative to the subtree to which the session is associated with. For example, if the session is opened on:

```
./Vendor
```

Then the following URI address the Bach ring tone:

```
RingTones/Bach
```

If the session is opened with a null URI, the following URI must be used to address the Bach ring tone.

```
./Vendor/RingTones/Bach
```

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 that can verify if a node is valid.

11.4.5 Creating Nodes

The methods that create interior nodes are:

- createInteriorNode(String) – Create a new node with no meta data.

From the OMA DM manual: If the principal does not have Replace access rights on the parent of the new node then the session must automatically set the ACL of the new node so that the creating server has Add, Delete and Replace rights on the new node.

- createInteriorNode(String,String) – Create a new node and derive the meta data from the DDF document that the second parameter, a string that can be interpreted as a URL, points to. The DDF type is described ####.

Why is the type not a URL?

- createLeafNode(String,DmtData) – creates a new leaf node. The value of the new node is defined by the DmtData object passed as the second parameter. If the default constructor for the DmtData object is used the node's default value must be used instead. If such default value does not exist, a Dmt Exception must be thrown.

Why not just null for the default?

If the MIME type of the given DmtData object is not set, it must be obtained from the corresponding Dmt Meta Node. If the meta node carries more than one MIME type a Dmt Exception must be thrown

What about other validation information in the meta node? Do we do a full check?

For a node to be created, the following conditions must be fulfilled:

- The parent of the node must exist, i.e. there is no automatic creation of intermediate nodes.
- The URI of the new node has to be valid

The principal of the Dmt Session must have ACL-level permission to add the node to the parent.????

- The caller must have the necessary Dmt Permissions.
- All constraints must be satisfied

yes?

11.4.6 Node Properties

A DMT node has a number of runtime properties that can be set through the session object. These properties are:

- *Title* – (String) A human readable title for the object. The title is distinct from the node name. The title can be set with setNodeTitle(String, String) and read with getNodeTitle(String).

How do we handle localization?

- *Type* –(String) The MIME type, as defined in [24] *MIME Media Types*, of the node's value.The type of interior node is an URL pointing to a OMA DM DDF. The type can be set with setNodeType(String,String) and read with getNodeType(String).
- *Version* – (int) Version number, incremented after every modification for both a leaf and an interior node. The getNodeVersion(String) method returns this version, the value is read-only.

Thus an interior node changes as well? Does it depend on its descendants?

- *Size* – (int) Number of bytes of a leaf node's data. Zero for interior nodes. The size is read-only and can be read with getNodeSize(String).
- *Time Stamp* –(Date) Time of last change. The getNodeTimestamp(String) returns the time stamp. The value is read only.
- *ACL* – The Access Control List for this and descendant nodes. The property can be set with setNodeAcl(String,DmtAcl) and obtained with getNodeAcl(String).
- *Value* – The data value of a leaf node (must be null for an interior node). The value is stored in a DmtData object. This object is discussed in the next section. It can be set with setNodeValue(String,DmtData) and obtained with getNodeValue(String).
  If the DmtData object does not specify a MIME type, the MIME type must remain unchanged.

11.4.7 Dmt Data

Values are represented as DmtData objects. These are objects that are dynamically typed by a DmtDataType enumeration. In OMA DM, this enumeration is called the *format*.of the data value.

Should this object be immutable?

If the constructor ### gone ### is called, an *empty* DmtData object is created. In that case, the node's format is UNSPECIFIED. This usually indicates that the default value, which depends on the context, must be used.

The format of the DmtData class is similar to the type of a variable in a programming language.

A getFormat that returns an enumeration of a DataType is a misnomer. Use Format or Type, but do not mix

Why is this enumeration not in the DmtData class?

What is the difference between NULL and UNSPECIFIED?

Formats are:

- UNSPECIFIED – The type when the empty constructor is used to create this DmtData object. There is no valid data.
- NULL – The data must be null, indicating there is no valid data available.
- BINARY – A byte array. The byte array can only be obtained with getBinary()
- BOOLEAN – A boolean, it can only be obtained with getBoolean().
- INTEGER – An int. Only the getInt() method returns this value.
- NODE –

I do not understand where this type could be used?

- STRING – A String, can only be obtained with getString().
- XML – A string containing an XML fragment. It can be obtained with getString().

Why is there no getXML and setXML?

Must the XML have <?xml ...?>? Must it be valid? Namespaces?

The DMT recognizes a MIME type that reflects how the data should be interpreted. For example, it is possible to store a GIF and a JPEG image in a DmtData object with a BINARY format. Both the GIF and the JPEG object share the same DmtDataType format, but will have MIME types of image/jpg and image/gif.

The DmtData class has two groups of constructors, one with an explicit MIME type, the other one without. The group without constructor sets the MIME type to null. The other group of constructors allow the MIME type to be defined.

It feels like hte MIME type should be part of the Node, not the dmt data. It feels very odd here.

The getString method is kind of weird in the implementation, it seems to allow all types? That is ok for toString() but getString() should be more restrictive imho

It stills very bad that even Monitoring and DMT have to develop their own dynamic typing :-(

Is the Dmt Admin doing any validation?

11.4.8 Deleting Nodes

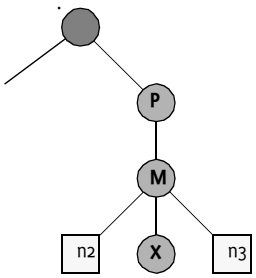
The deleteNode(String) method on the session represents the Delete operation. It deletes a node and all its descendants. This method is applicable to leaf and interior nodes.

Is there any automatic checking of the referential integrity?

Is there a noticaeable order in deletion?

For example, given Figure 42, deleting node P must delete the nodes ./P„./P/M, ./P/M/X, ./P/M/n2 and ./P/M/n3.

Figure 42 *DMT node and deletiion*



11.4.9 Copying Nodes

The copy(String,String,boolean) method on the DmtSession object represents the Copy operation. A node is completely copied to a new URI. It can be specified with a boolan if the whole subtree (true) is copied or just the indicated node.

The copy operation is logically treated as a number of Add and Replace operations. There is no separate operation on the ACLs for Copy.

Is the ACL also copied?

11.4.10 Renaming Nodes

The renameNode(String,String) method on the DmtSession object represents the Rename operation. A node's name is changed.

The Rename operation is logically treated as a number of Add and Delete operations.

How is the rename operation secured with ACLs? Delete-Add?

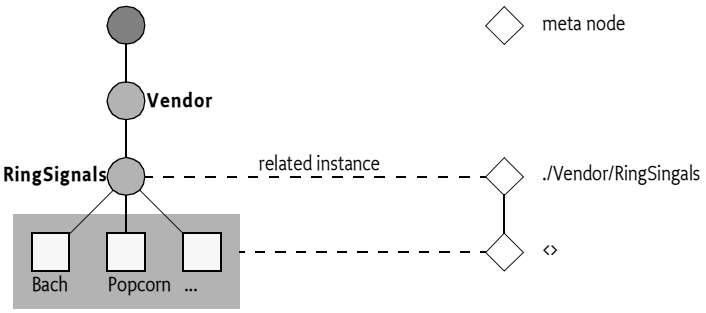
How about referential integrity constraints?

11.5 Meta Data

The session object can provide meta-data, which is data about the capabilities of a target node. The ###gone### method returns a DmtMetaNode object for a given URI. This node is called the *meta node*. A meta node can exist without an actual instance being present. This model is depicted in Figure 43.

For example, if a new ring tone, Grieg was created in Figure 43 on page 209 it would be possible to get the Meta Node for ./Vendor/RingSignals/Grieg before the node was created.

Figure 43 *Nodes and meta nodes*



A node can optionally have a meta node associated with it. The one or more nodes that are described by the meta nodes are called the meta node's *related instances*.

A meta node of a node can change once it is created or deleted. There is no guarantee that the ###getMetaNode(String)### always returns the same meta node for the same URI.

where does the meta data come from? Is this always the plugin?

The DmtMetaNode interface supports methods to retrieve read-only meta data; both standard OMA DM as well as defined OSGi extensions. The extension is to provide for better DMT data quality in an environment where many software components manipulate this data. This extension does not break compatibility with OMA DM. Compatibility with OMA DM is further discussed in *Differences with OMA DM* on page 212.

The getChildURIs and getDependentURIs are, in a way, not read only.

What is the risk if OMA DM goes into another direction?

Please verify this table ...

	NODE	INTEGER	BINARY	BOOLEAN	STRING	UNSPECIFIED
canAdd	x					
canDelete, canExecute	x	x	x	x	x	
,canGet, canReplace	x					
isPermanent	x	x	x	x	x	
getDefault		x	x	x	x	
getDescription	x	x	x	x	x	
getMax, getMin, hasMax, hasMin		x				
getMimeTypes, getDependentURIs, getChildURIs		x	x	x	x	
getReferredURI					x	
getMaxOccurrences, getZeroOccurrencesAllowed	x	x	x	x	x	
getValidValues		x	x	x	x	
getRegExp					x	

Table 11 *Matrix of meta data versus format of node.*

11.5.1 Operations

The ####canAdd()### methods provide information if the associated node can perform the related operation. For example, if the canExecute method returns true, the target object supports the Execute operation. I.e. calling the execute(String,String) method with the target URI is possible. However, permissions can still forbid the successful execution. The meta data must be independent of the permissions.

For example:

```
void foo( DmtSession session, String nodeUri ) {
  if ( session.getMetaNode(nodeUri).canExecute() (
      session.execute(nodeUri,"foo" );
}
```

Normally we do these methods with a parameter so it can be easier extended in the future: can("ADD") or so. Adding methods to interfaces is not backward compatible.

11.5.2 Miscellaneous

- getScope() – (boolean) Certain nodes represent structures in the devices that can just never be deleted, they are always present. For example a node representing the battery level can never be deleted because it is an intrinsic part of the device.

Shouldn't this be called getScope and return an int to be compatible with OMA DM? If they add a new scope, we have to add a method ..

- getDescription() – (String) A description of the node. Descriptions can be used in dialogs with end users, e.g. a GUI application that allows the user to set the value of a node.

How is localization handled?

- getDefault() – (DmtData) A default data value.

11.5.3 Validation

The validation information allows the runtime system to verify constraints on the values, however, it also allows user interfaces to provide guidance.

where is applied or is it just info?

11.5.3.1 Data Types

A node can be constrained to a certain format and one of a set of MIME types.

- getFormat() – (int) The required type. This is an enumeration from the DmtDataType interface. See *Dmt Data* on page 206 for further information. It returns UNSPECIFIED format if the node can take one of multiple possible formats at run time.
- getMimeTypes() – (String[]) A list of MIME types. If this is null, the DmtData value object can hold any arbitrary MIME type. Otherwise, the MIME type of the given DmtData object must be a member of the list returned from the getMimeTypes method.

11.5.3.2 Cardinality

A meta node can constrain the number of siblings. This can be used to verify that a node must not be deleted because there should be at least one node left (isZeroOccurrenceAllowed()), or to verify that a node cannot be created because there are already too many siblings (getMaxOccurrence()).

I am confused what happens when you have multiple types on a node. Is the cardinality just the # of children of the parent, or do you have to verify that the children are really the correct type? If I have a node of type X and it has 2 types of children that are associated with meta node MA and MB. Does the cardinality count for ALL of the children of X or only the ones that have the same meta node? It feels like the cardinality should be specified by the parent?

For example, the ./Vendor/RingSignals/<> meta node could specify that there should be between 0 and 12 ring signals.

- getMaxOccurrence() – (int) A value greater than 1 that specifies the maximum number of instances for this node.
- isZeroOccurrenceAllowed() – (boolean) Returns true if zero instances are allowed. If not, the last instance must not be deleted.

11.5.3.3 Matching

The following methods provide matching capabilities.

- getRegExp() – (String) A regular expression that must follow the syntax for regular expressions used with the java.util.regex.Pattern class. Regular expressions are a convenient way to test the structure of a string.

Should be called getRegularExpression or getPattern.

Is this also valid for non string values.

- getValidValues() – (DmtData[]) A set of possible values for a node, or null otherwise. This can for example be used to give a user a set options to choose from.

11.5.3.4 Integer Ranges

Integer nodes (format must be INTEGER) can be checked for a minimum and maximum value. These constraints are optional, the presence of these constraints can be tested for.

Minimum and maximum values are inclusive. I.e. the range is [getMin(), getMax]. For example if the maximum value is 5 and the minimum value is -5 then the range is [-5,5]. This means that valid values are -5,-4,-3,-2 ... 4, 5.

yes?

What happens with these methods when the format is not INTEGER?

- hasMax() – (boolean) If true, a valid constraint is available with the getMax method. Otherwise, the getMax method returns Integer.MAX_VALUE.
- getMax() – (int) The value of the node must be less or equal to this maximum value.

Why do we need the hasXXX methods? It seems to have very little value. Same for hasMin

- hasMin() – (boolean) If true, a valid constraint is available with the getMin method. Otherwise, the getMin method returns Integer.MIN_VALUE.
- getMin() – (int) The value of the node must be greater or equal to this minimum value.

11.5.4 Differences with OMA DM

As the meta data of a node in OSGi is richer than what is mandated by OMA DM, the Dmt Admin nodes can not be fully described by OMA DM's DDF (Device Description Framework). How the management server learns the OSGi management object structure is out of the scope of this specification.

We learned with meta type that was a big mistake ...

The following table shows the differences between the OSGi meta data and the Data Description Format of the OMA. The DTD description of DDF can be found at [17] *SyncML Device Management Tree and Description, Version 1.1.2*.

Please verify this table ...

	DDF Fragment	Comment
canAdd,	AccessType: Add	
canDelete	AccessType: Delete	
canExecute	AccessType: Exec	
canGet	AccessType: Get	
canReplace	AccessType: Replace	
	AccessType: Copy	Missing in OSGi??
isPermanent	Scope: Permanent | Dynamic	
getDefault	DefaultValue:	
getDescription	Description:	
getMax, getMin, hasMax, hasMin		Missing in OMA
getMimeTypes	Type: MIME List	OSGi allows multiple MIME Types
getDependentURIs, getChildURIs, getReferredURI		Absent in OMA
getMaxOccurrences, getZeroOccurrencesAllowed	Occurrence: One | ZeroOrOne | ZeroOrMore | OneOrMore | ZeroOrN | OneOrN	
getValidValues		Absent in OMA
getRegExp		Absent in OMA

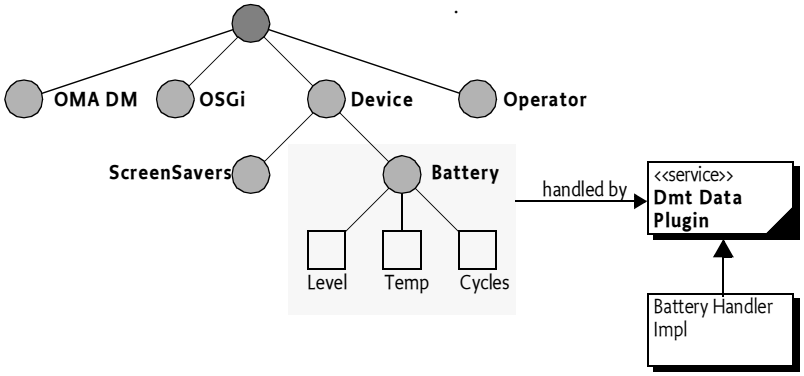
Table 12 *Comparison of OMA DM DDF versus OSGi meta data*

11.6 Plugins

The Plugins take the responsibility of handling DMT operations within certain subtrees of the DMT. It is the responsibility of the Dmt Admin service to forward the operation requests to the appropriate plugin. The only exception is the ACL manipulation commands. ACLs are enforced by the Dmt Admin service level and not by the plugin, security related commands are therefore not forwarded to plugins.

Confused, where do the ACLs get checked

Figure 44 *Device Management Tree example*



Does this mean plugins are optional? I think that a compliant impl. must support plugins? The plugins are as much part of the contract as the Dmt Admin interface

Plugins are OSGi services, the Dmt Admin must dynamically add and remove the plugins as they are registered. Service properties are used to specify the subtree that the plugin can manage.

For example, a plugin related to Configuration Admin handles the subtree which stores configuration data. This subtree could start at ./OSGi/cfg. When the client wants to add a new configuration object to the DMT, it must issue an Add operation to the ./OSGi/cfg node. The Dmt Admin forwards this operation to the configuration plugin. The plugin maps the request to one or method calls on the Configuration Admin service. Such a plugin is a simple proxy to the Configuration Admin service, it provides a DMT view of the configuration data store.

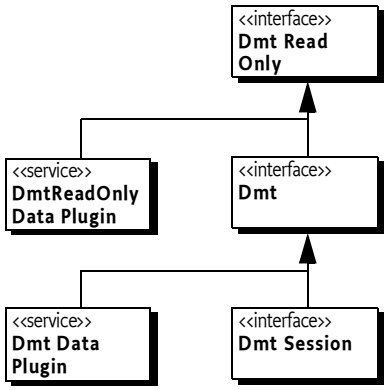
There are two type of Dmt plugins: *data plugins* and *exec plugins*. A data plugin is responsible for handling the node retrieval, addition and deletion operations, while an exec plugin handles the node execution operation.

Data plugins come in two flavours:

- DmtReadOnlyDataPlugin – This type of plugin supports only data *retrieval*.
- DmtDataPlugin – This plugin type supports all methods of the DmtReadOnlyDataPlugin interface, but additionally supports the modification of node values and the tree structure.

The data plugins contain most of the methods that the Dmt Session does. The Dmt Read Only Data Plugin service inherits these methods from the DmtReadOnly interface, the Dmt Data Plugin service inherits from the Dmt interface (which in it turn inherits from DmtReadOnly interface). The different interfaces makes extensive use of interface inheritance. Figure 45 shows this inheritance model.

Figure 45 *Inheritance Diagram of key interfaces*



It i skind of odd that DmtDataPlugin does not inherit DmtReadOnlyDataPlugin ... The whole inheritance is kind of awkward imho

11.6.1 Associating a Subtree

Each plugin is associated with one ore more DMT subtrees. The plugin - subtree association is called the *plugin root*. The plugin root is defined by a service registration property. This property is different for exec plugins and data plugins:

- `dataRootURIs` – (`String[]`, `String`) Must be used by data plugins.
- `execRootURIs` – (`String[]`, `String`) Must be used by exec plugins.

Why are they different?

For example, a data plugin can register itself in its activator to handle the subtree `./Dev/Battery`:

```
public void start(BundleContext context) {
  Hashtable ht = new Hashtable();
  ht.put(Constants.SERVICE_PID, "com.acme.data.plugin");
  ht.put( DmtDataPlugin.DATA_ROOT_URIS, "./Dev/Battery");
  context.registerService(
      DmtDataPlugin.class.getName(),
      new BatteryHandler(context);
      ht );
}
```

If this activator was executed, an access to `./Dev/Battery` must be forwarded by the Dmt Admin service to this plugin.

11.6.2 Overlapping Subtrees

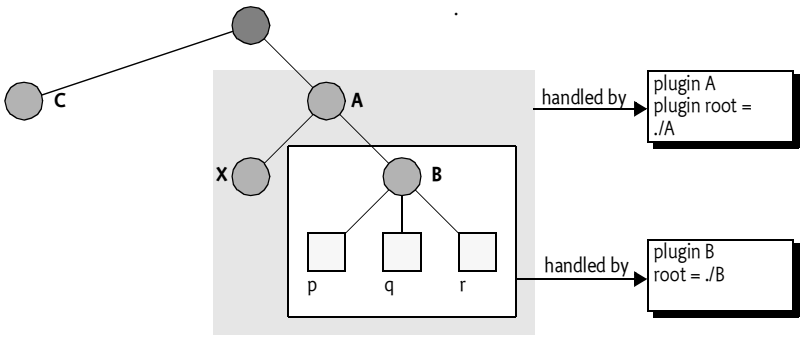
Overlapping plugins are explicitly allowed. It is allowed for a plugin A to have plugin root `./A` while a plugin B has the plugin root `./A/B`.

If a URI addresses multiple plugins due to such an overlap, then the plugin that has the longest plugin root must be chosen. This plugin is the one which is the closest in the tree to the requested URI.

What happens if the plugin roots are equal?

In the previous example, the URI ./A/B/C must be handled by plugin B, the URI ./A/X must be handled by plugin A.

Figure 46 *Device Management Tree example*



This looks like a scary relationship. Assume B has an error and is not registered and A is some database plugin (which I assume is the idea). Then when B comes up later, it hides potentially relevant information. Suggest to not allow overlap.

11.6.3 Plugin Meta Data

Meta data can be supported in a proprietary way by the Dmt Admin service. However, plugins can optionally provide meta data. For example, when a data plugin is implemented on top of a data store that has its own meta data facility. This way redundancy and inconsistencies can be avoided.

This sounds like it is an exceptional case. This is different from what I would have expected. It seems logical that the unit that for example provides the battery interface also provides the meta data? Otherwise you create a lot of communication during development?

Meta data provided by the plugin must take precedence over whatever meta data the Dmt Admin service has for a specific node.

If both an exec plugin and data plugin register at the same plugin root, who delivers the meta data?

If a node does not support the meta data, could an overlapping plugin provide it? I.e. same rules for meta nodes as for nodes?

11.6.4 Plugins and Transactions

For the Dmt Admin service to be transactional, it requires that transactions are supported by the data plugins. This is not mandatory in this specification and the Dmt Admin service therefore has no transactional guarantees for atomicity, consistency, isolation and durability.

However, the DmtAdmin interface and the DataPlugin interfaces are designed to support Dmt Data Plugin services that are transactional.

Exec plugins need not be transaction aware because the execute method does not provide transactional semantics.

What is the reason? (I added one). This stuff needs to be mentioned in the Javadoc

The Dmt Admin must guarantee that a plugin, at any moment in time, participates at most in a single session to create an isolation between sessions. When a plugin is used for the first time in a session, and is not locked by another session, the Dmt Admin must call the open method on the plugin.

When a session is closed, the Dmt Admin service must call the close method on all plugins that had their open method called during the life time of the session, a.k.a. the participants. If the plugins support transactionality and the session is atomic, the close method is the place for a plugin to commit if it had outstanding changes. I.e. after the close method has returned, changes made during the session must be persistent.

Do we define an ordering? I.e. reverse order of open?

Why does the read only have a close method?

If a atomic session is rolled back (rollback method called), the Dmt Admin service must call the rollback method of all Dmt Data Plugin services that participated in the session. Dmt Data Plugin services should at that time undo any changes that were made after their open method was called.

Note the "should undo". Or are we harder and do we have a MUST? I.e. must a data plugin provide full undo?

What does rollback mean for a non-atomic session?

How do we handle it when a plugin is unregistered inside a session?

Can we have a serious error that causes an automatic rollback?

11.6.5 Lack Of Two Phase Commit

The current design does *not* support the industry standard two-phase commit protocol. This can lead to serious consistency errors. For example, plugin A and plugin B participate in a session. When the session is committed, plugin A commits successfully, but plugin B encounters an error and rolls back. Now the whole session should be rolled back but plugin A has already committed its changes.

So what do we do in those cases? It feels rather poor.this way

11.7 Access Control Lists

Each node in the DMT can be protected with an *access control list.*, or *ACL.* An ACL is a list of associations between:

- *Principal* – The identity that is authorized to use the associated operations.
- *Operation* – A list of operations: ADD, DELETE, GET, REPLACE,.EXECUTE.

DMT ACLs are defined as strings with an internal syntax in [17] *SyncML Device Management Tree and Description, Version 1.1.2*. Instances of this class can be created by supplying a valid OMA DM ACL string as its parameter. The syntax of the ACL is presented here in shortened form for convenience:

```
acl        ::= ( acl-entry ( '&' acl-entry )* )?
acl-entry  ::= command '=' ( servers | '*' )
servers    ::=  server ( '+' server )*
server     ::= [^=&*+ \t]+
command    ::= 'Add' | 'Delete' | 'Exec' | 'Get' | 'Replace'
```

Examples:

```
Add=*&Replace=*&Get=*

Add=www.sonera.fi-8765&Delete=www.sonera.fi-
8765&Replace=www.sonera.fi-8765+321_ibm.com&Get=*
```

The DmtAcl(String) constructor can be used to construct an ACL from an ACL string. The toString() method returns a String object that is formatted in the specified form, also called the canonical form.

Is there a guarantee in order for principals + commands? I.e. is the canonical form guaranteed to be the same for two DmtAcls, regardless if they are constructed in totally different ways?

ACLs must *not* be verified by the Dmt Admin service. The Dmt Admin service must treat the ACLs as normal properties. The protocol adapter should verify ACLs.

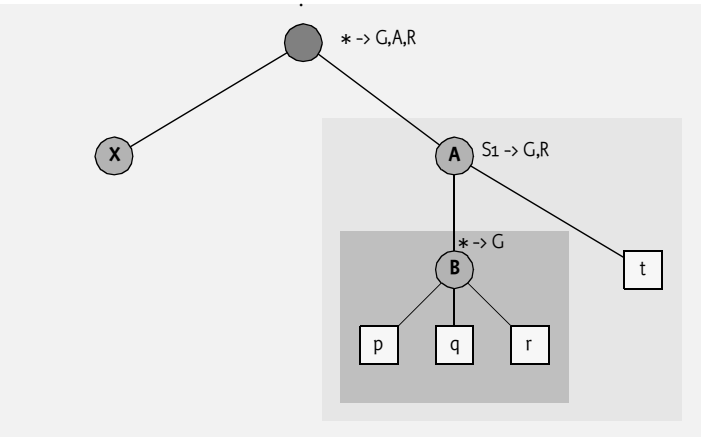
This is in the RFC, but I do not understand. This seems to imply that the Dmt Admin service also checks? It also seems to be not completely consistent.

Informative note. There are alternative solutions for checking the ACL, the specification only mandates that step 8 must not take place if the ACLs do not allow the execution of the management command. The sequence given above assumes that the protocol adaptor checks the ACL, which is how it is likely to happen in the case of a native protocol adaptor (a native management client). Another possible way is that after step 6, the management command is forwarded to the DmtSession without checking the ACL first, and the DmtSession contacts the DmtAdmin for checking the ACL.

ACLs are a property of a node. If this ACL is *empty*, i.e. contains no commands nor principals, then the *effective* ACL of that node must be the ACL of its first ancestor that has a non-empty ACL. This effect is shown in Figure 47. This diagram shows the ACLs set on a node and their effect (which is shown by the shaded rectangles). E.g. any server can get the value of `p`, `q` and `r`. However, only server `S1` is authorized to replace their value. Node `t` can only be read and replaced by server `S1`.

Node `X` is fully accessible to any authenticated server because the root node specifies that all servers have `Get`, `Add` and `Replace` access (`*->G,A,R`).

Figure 47 *ACL inheritance (letters are the first letter of the ACL commands)*



The definition and example demonstrated the access rights to the properties of a node. The ACL property has different rules. If a server has Replace access to a node, the server is permitted to change the ACL of all its nodes, regardless of the ACLs that are set on the child nodes.

yes????? This is very described very badly in the DMT spec. Ihope I got it right

In the previous example, only server S1 is authorized to change the ACL of node B because it has Replace permission on node B's parent node A.

Additionally, *for interior nodes only*, the ACL property of a node is guarded by its Replace permission. I.e. the ACL of an *interior* node can permit a server to change its ACL.

Figure 48 *ACLs for the ACL property*

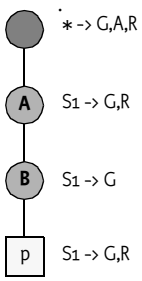


Figure 48 demonstrates the effect of this rule with an example. Server S1 can modify all nodes. A more detailed analysis:

- *Root* – The root allows all authenticated servers to access it. The root is an interior node so the Replace permission permits the change of the ACL property.
- *Node A* – Server S1 has Replace permission and node A is an interior node so server S1 can modify the ACL.
- *Node B* – Server S1 has no Replace permission for node B, however, the parent node A of node B grants server S1 Replace permission and S1 is therefore permitted to change the ACL.

- *Node t* – Server S1 must not be allowed to change the ACL of node t, despite the fact that it has Replace permission on node t. For leaf nodes permission to change its ACL is defined by the Replace permission in the parent node's ACL. This parent, node B, has no such permission set and thus, access is denied.

What a mess .... It looked like they loved the symmetry and then found out it did not work ...

The following methods provide access to the ACL property of the node.

- getNodeAcl(String) – Return the current ACL for the given node. The ACL may be empty if no ACL is set. I.e. no inheritance is taking into account, the effective permission is the responsibility of the caller. The returned DmtAcl object is a copy of the information. Changes to this DmtAcl object must not be reflected in the node's ACL.

Is this taking the inheritance into account? I guess not, but that should be spelled out?

Can it return null? If not, shouldn't it have an isEmpty method?

- setNodeAcl(String,DmtAcl) – Set the node's ACL. The ACL can be empty in which case the effective permission must derived from an ancestor. The Dmt Admin service must copy the information from the given DmtAcl object. Changes to the given DmtAcl object must not be reflected in the node's ACL after this method has returned.

The DmtAcl class maintains the permissions for a given principal in a bit mask. The following permission masks are defined as constants in the DmtAcl class:

- ADD
- DELETE
- EXEC
- GET
- REPLACE

Bit masks allow for fast checking and altering. The DmtAcl class provides a number of methods that incrementally build the ACL using the masks. The class features methods for getting, setting and modifying permissions for given principals.

The DmtAcl class is not immutable. How much threat is there that someone changes the instance after it has been given as a parameter? Shouldn't this class become immutable? So far in the OSGi we also have been very careful not to have mutable classes ...

- addPermission(String,int) – For the given principal, add the given permissions. I.e. the given permission mask is OR'ed with the existing permission mask.
- deletePermission(String,int) – For the given principal, remove the given permissions. I.e. the given permission mask negated and then AND'ed with the existing permission mask.
- setPermission(String,int) – For the given principal, replace the permission mask with the given permission mask.

Information from a given ACL can be retrieved with

- getPermissions(String) – Return the combined permission mask for this principal.
- getPrincipals() – (Vector)Return an list of principals (String objects)

In OSGi style this method should return an array, not a vector

Additionally, the isPermitted(String,int) method verifies if the given ACL authorizes the given permission mask. The method returns true if all commands in the mask are allowed by the ACL. For example:

```
DmtAclacl = new DmtAcl("Get=S1&Replace=S1");

if (  acl.isPermitted("S1", DmtAcl.GET+DmtAcl.REPLACE ))
   ... // will execute

if ( acl.isPermitted(
   "S1", DmtAcl.GET+DmtAcl.REPLACE+DmtAcl.ADD ))
   ... // will NOT execute
```

11.7.1 Ghost ACLs

The ACLs are fully maintained and enforced by the Dmt Admin service, the plugin must be unaware of the ACLs. The Dmt Admin service must synchronize the ACLs with any change in the DMT that is made through its service interface. For example, if a node is deleted through the Dmt Admin service, it must also delete an associated ACL.

Dmt service, enforces ACLs?

However, the DMT nodes are mapped to plugins and plugins can delete nodes outside the scope of the Dmt Admin service. I.e. a node can be deleted without the Dmt Admin being aware of it.

As an example, consider a configuration dictionary which is mapped to a DMT node that has an ACL. If the configuration dictionary is deleted using the Configuration Admin service, the data disappears but the ACL entry in the Dmt Admin remains. Even worse, if the configuration dictionary is recreated with the same PID, it will get the old ACL, which is not the intended behavior.

This specification does not specify a solution to solve this problem. Implementation specific work-arounds should be used:

- Use a proprietary callback mechanism from the data back-end to notify the Dmt Admin service to clean up the ACLs.
- Implement the services on top of the DMT. For example, the Configuration Admin service could use a plugin that provides general data storage service.

11.8 Notifying the Server

In certain cases it is necessary for some code on the device to notify a remote management server, this is called an *alert*. Alerts can for example be necessary to:

- A plugin that must send the result of an asynchronous EXEC operation. This for example used in the to return data from a device log query and for reporting KPI changes.
- Sending a session request to the server.
- Notifying the server of completion of a software update operation.

Notifications can be send to a management server using the Dmt Alert Sender service. This service is typically registered by the Dmt Admin service implementation. The remote server is notified with one or more DmtAlertItem objects. The DmtAlertItem describes details of various alerts that can be sent by, for example, the client of the OMA DM protocol.

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.

The DmtAlertItem class contains the following items:

- source – (String) The URI of a DMT node that is related to this request. This parameter can be null.

Why is the source in the DmtAlertItem?? Isnt this a part of the send operation? If you send multiple items, should each item have its own source?

- type – (String)

I assume this is the MIME type?? Why is the Alert Item then not a DmtData? That seems to hold all the relevant info type, format, data.

- format – (String)

Is this the node format? If so, why isnt it an int? What should be used here?

- mark – (String) The DmtAlertItem class can automatically create an appropriately formatted XML block for a data item, given the type, format and data. However, in certain cases the notification requires specially formatted XML. In those cases, the markup can be set explicitly.

The implementation is painfully inefficient and does not escape XML entitities

Shouldnt this not be called xml or markup?

Generating the XML is the responsibility of the Dmt Admin service, this kind of encoding does not belong here.

- data – (String) The payload of the notification.

A DmtAlertItem object can be constructed with one constructor that sets all fields:

- DmtAlertItem(String,String,String,String,String) – This method takes all the previously defined fields.

The whole DmtAlertItem seems to be very similar to a DmtData object ... Why not reuse it? A DmtData can also handle the XML/mark case ... Much simpler.

In OMA DM, sending an alert requires an *alert code*. Alert codes are defined by ####. An alert code is a type identifier for the notification, it usually requires specifically formatted DmtAlertItem objects.

where?

The Dmt Alert Sender service provides the following methods to send DmtAlertItem objects to the management server:

- sendAlert(DmtSession,int,DmtAlertItem[]) – Send the alert to the server that is associated with the session.

Why not put this method on the session?

- sendAlert(String,int,DmtAlertItem[])
- sendAlert(int,DmtAlertItem[])

Why are all these methods not grouped together

sendAlert(String principal, int code, DmtData[])

, with a null when you do not know the server id. The session has a getPrincipal() method so it is trivial to get the principal

All forms feature an array of alert items to be sent and an alert code associated with the alert. It is the Dmt Admin's responsibility to route the alert to the appropriate protocol adaptor. This routing is implementation dependent, there is no mechanism specified to influence this routing.

Implementers should base the routing on the session or server information provided as a parameter in the sendAlert methods. If the client can provide the session or server information, it should do so. Routing might even be possible without any routing information if there is a well known remote server for the device.

If the request cannot be routed, the Dmt Alert Sender service must throw an Dmt Exception with a code of ALERT_NOT_ROUTED.

What kind of guarantees do we get for delivery? I.e. should the sender retry?

It is not clear to me why the sendAlert is not on the Dmt Admin service?

11.9 Exceptions

Most of the methods of this Dmt Admin service API throw Dmt Exceptions whenever something goes wrong. The DmtException class contains numeric error codes which describe the cause of the error. Some of the error codes correspond to the codes described by the OMA DM spec, some are introduced by the OSGi Alliance.

You need a test to easily discriminate them, i.e. OSGi codes are negative?

All possible error codes are constants in the DmtException class:

- ALERT_NOT_ROUTED – The sendAlert method could not route the alert to the remote server.
- COMMAND_FAILED –
- COMMAND_NOT_ALLOWED –
- CONCURRENT_ACCESS –
- DATA_STORE_FAILURE –

- ###DEVICE_FULL### –
- FEATURE_NOT_SUPPORTED –
- FORMAT_NOT_SUPPORTED –
- METADATA_MISMATCH –
- NODE_ALREADY_EXISTS –
- NODE_NOT_FOUND –
- OTHER_ERROR –
- PERMISSION_DENIED –
- REMOTE_ERROR –
- ROLLBACK_FAILED –
- PERMISSION_DENIED?
- URI_TOO_LONG –

I need to know for each error, when and where it can be raised.

11.10 Events

The Dmt Admin service does not have a special Dmt Listener, instead it uses the Event Admin service. Events are only send when a session is closed. Depending on the operations, the following events must be send in arbitrary order:

If we do not have ordering, we make testing harder and it can easily create incompatibilities. I suggest we add an ordering in event types + URIs in the arrays

- org.osgi.service.dmt.DmtEvent.ADDED – New nodes were added.
- org.osgi.service.dmt.DmtEvent.DELETED – Existing nodes were removed.
- org.osgi.service.dmt.DmtEvent.REPLACED – Existing node values or other properties were changed.
- org.osgi.service.dmt.DmtEvent.RENAMED – Existing nodes were renamed.
- org.osgi.service.dmt.DmtEvent.COPIED – Existing nodes were copied.

existing? What does it mean when a session ADDs and REPLACE? Is that an existing node that is REPLACED?

What happens when a node is added and deleted?

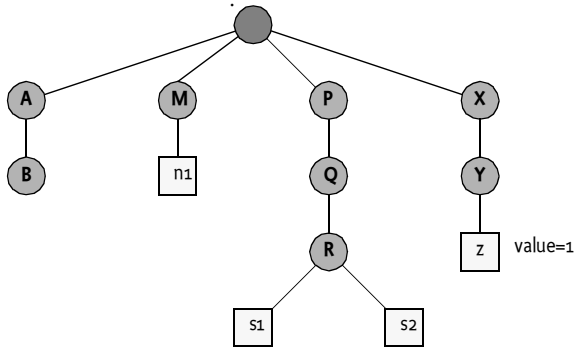
DMT events have the following properties:

- session.id – (Integer) A unique identifier for the session that triggered the event. This property has the same as getSessionId() of the associated session.
- nodes – (String[]) The absolute URIs of each affected node. This is the nodeUri parameter of the Dmt API methods.
- newnodes – (String[]) – The absolute new URIs of renamed or copied nodes. the Only the RENAMED and COPIED events have this property. The newnodes array runs parallel to the nodes array. In case of a rename newnodes[i] must contains the new name of nodes[i], and in case of a copy, newnodes[i] is the URI where nodes[i] was copied to.

Does a COPY also have a ADDED?

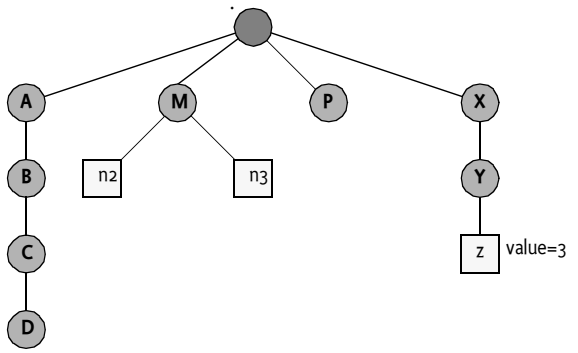
For example, in a given session, given the DMT in Figure 49 is modified with the following operations:

Figure 49 *Example DMT before*



- Add node ./A/B/C
- Add node ./A/B/C/D
- Rename ./M/n1 to./M/n2
- Copy ./M/n2 to ./M/n3
- Delete node ./P/Q
- Replace ./X/Y/z with 3

Figure 50 *Example DMT after*



When the close method of the Dmt Session is called, the following events are published by the DMT in an arbitrary order:

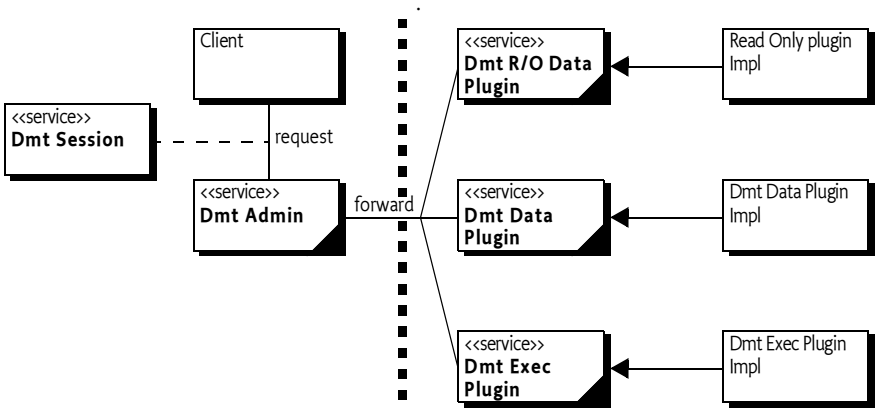
```
org.osgi.service.dmt.DmtEvent.ADDED {
      nodes         = [./A/B/C, ./A/B/C/D ]
      session.id    = 42
}
org.osgi.service.dmt.DmtEvent.REPLACED {
      nodes         = [ ./X/Y/z ]
      session.id    = 42
}
org.osgi.service.dmt.DmtEvent.REMOVED {
      nodes         = [ ./P/Q, P/Q/R, P/Q/R/s1, P/Q/R/s2 ]
      session.id    = 42
```

```
}
org.osgi.service.dmt.DmtEvent.RENAMED {
      nodes         = [ ./M/n1 ]
      newnodes      = [ ./M/n2 ]
      session.id    = 42
}
org.osgi.service.dmt.DmtEvent.COPIED {
   nodes            = [ ./M/n2 ]
   newnodes         = [ ./M/n3 ]
   session.id       = 42
}
```

11.11 Security

A key aspect of the Dmt Admin service model is the separation from DMT clients and plugins. The Dmt Admin service receives all the operation requests and, after verification of authority, forwards the requests to plugins.

Figure 51 *Separation of clients and plugins*

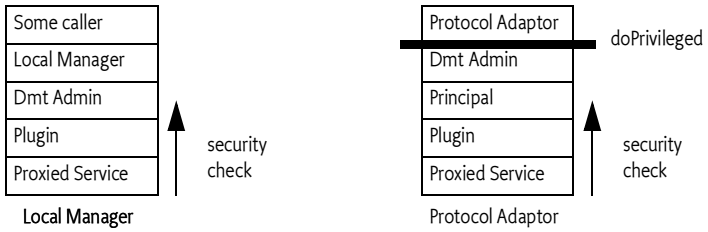


This architecture makes it straightforward to use the OSGi security architecture to protect the different actors.

11.11.1 Operational Permissions

There is a distinct difference between the operational security of a local manager and a remote manager. The distinction is made on the principal. Protocol adapters should use the getSession method that takes an authenticated principal.Local managers should not specify a principal.

Figure 52 *Access control context, for local manager and protocol adapter operation*



11.11.1.1 Protocol Adapters

A protocol adapter must provide a principal to the Dmt Admin service when it gets a session, i.e. it must use the getSession(String,String,int) method. The protocol adapter must have Dmt Principal Permission for the given principal. The Dmt Admin must then use this principal to determine the *security scope* of the given principal. This security scope is a set of permissions. This set can come from the Conditional Permission Admin service or through some proprietary way. For example, the principal name could be related to a certificate signer tha Conditional Permission Admin can relate to a security scope.

The Dmt Admin service does not perform any ACL checking. ACL checking must be performed by the protocol adapter.

another place unclear with ACL checking

Any operation that is requested by a protocol adapter must be executed in a doPrivileged block that takes the principal's security scope. The doPrivileged block effectively hides the permissions of the protocol adapter, all operations must be performed under the security scope of the principal.

The security check for a protocol adapter is therefore as follows:

- The operation method calls doPrivileged with the security scope of the principal.
- The checkPermission method is called with a Dmt Permission for the given node URI and operation. A thrown Security Exception must be passed unmodified to the caller of the operation method.
- The operation is forwarded to the appropriate plugin. The plugin can perform additional security checks. See ###

The stack is shown in Figure 52 with the protocol adapter stack column.

This principal based security model allows for minimal permissions on the remote adapter. However, this model does not guard against malicious protocol adapters. The protocol adapter is responsible for the authentication of the principal. Once it has permission to use that principal, it can at any time use any DMT command that is permitted for that principal.

I wonder if it is not just easier to let the protocol handler do the doPrivileged? Would probably simplify implementations. There seems to be no real security advantage in this model.

sequence diagram

11.11.1.2 Local Manager

A local manager does not specify a principal. Security checks are therefore performed against the security scope of the local manager bundle. This is show in Figure 52 with the local manager stack. An operation is checked only with a Dmt Permission for the given node URI and operation. A thrown Security Exception must be passed unmodified to the caller of the operation method.

A local manager, and all his callers, must therefore have sufficient permission to handle the DMT operations as well as the permissions required by the plugins when they proxy other services.

sequence diagram

11.11.2 Plugin Security

Plugins are required to hold the maximum security scope for any services that they proxy. For example, the plugin that manages the Configuration Admin service must have ConfigurationPermission("*","*") to be effective.

I wonder if we should not mandate the plugins to do a doPrivileged as well. The current model is hardly manageable.

11.11.3 Dmt Principal Permission

Execution of the getSession methods of the Dmt Admin service featuring an explicit principal name is guarded by the Dmt Principal permission. This permission must be granted only to protocol adaptors who open Dmt Sessions on behalf of remote management servers.

This class is not ready yet ...

The DmtPrincipal class has no defined actions. The target is the URI names that can include a final wildcard character to match the target URI as a prefix.

Example:

```
new DmtPrincipalPermission("www.acme.com-*", "" )
```

11.11.4 Dmt Permission

The Dmt Permission controls access to management objects in the DMT. It is intended to control only the *local* access to the DMT. The Dmt Permission target string identifies the target node's URI (absolute path is required) and the action field lists the management commands that are permitted on the node.

The URI can end in a wildcard character * to indicate it is a prefix that must be matched. This comparison is string based so that node boundaries can be ignored.

Is this on node boundary or string based? I.e. ./De* matches ./Dev/battery? or is only ./Dev/* allowed? Further, do we support the distinction between ./Dev/- and ./Dev/* which dinstinguishes between 1 level and recursive sublevels.?

The following actions are defined:

- Add
- Delete
- Get
- Replace
- Execute

For example, the following code creates a Dmt Permission to check if the callers can add a new node to the node with the given URI.

```
new DmtPermission("./D*", "Add,Replace")
```

This permission must imply the following permission:

```
new DmtPermission("./Dev/Battery/level", "Replace")
```

There are no constants in the permission classes

11.11.5 Security Summary

11.11.5.1 Dmt Admin Service

The Dmt Admin service is likely to require All Permission. This is caused by the plugin model. Any permission required by any of the plugins must be granted to the Dmt Admin service. This is a large and hard to define set. However, the following list shows the minimum permissions required if the plugin permissions are left out.

```
ServicePermission    ..DmtAdmin                REGISTER
ServicePermission    ..DmtAlertSender          REGISTER
ServicePermission    ..DmtReadOnlyDataPlugin   GET
ServicePermission    ..DmtDataPlugin           GET
ServicePermission    ..DmtExecPlugin           GET
ServicePermission    ..EventAdmin              GET
DmtPermission        *                         *
DmtPrincipal
  Permission         *
PackagePermission    org.osgi.service.dmt      EXPORT
```

11.11.5.2 Dmt <X> Plugin

```
ServicePermission    ..DmtAdmin                GET
ServicePermission    ..DmtAlertSender          GET
ServicePermission    ..Dmt<X>Plugin            REGISTER
PackagePermission    org.osgi.service.dmt      IMPORT
```

11.11.5.3 Local Manager

```
ServicePermission    ..DmtAdmin                GET
ServicePermission    ..DmtAlertSender          GET
PackagePermission    org.osgi.service.dmt      IMPORT
DmtPermission        <scope>                   ...
```

Additionally, the local manager requires all permissions that are needed by the plugins it addresses.

11.11.5.4 Protocol Adapter

The Protocol adapter only requires Dmt Principal Permission for the instances that it is permitted to manage. The other permissions are taken from the security scope of the principal.

```
ServicePermission     ..DmtAdmin                GET
ServicePermission     ..DmtAlertSender          GET
PackagePermission     org.osgi.service.dmt      IMPORT
DmtPrincipalPermission<scope>
```

11.12 org.osgi.service.dmt

Invalid tag on top level code (x=19,y=3) [p.253]

This interface collects basic DMT operations. The application programmers use these methods when they are interacting with a <code>DmtSession</code> which inherits from this interface. The <code>DmtDataPlugin</code> and the </code>DmtReadOnlyDataPlugin</code> interfaces also extend this interface. <p> If the admin or a data plugin does not support an optional DMT property (like <code>timestamp</code>) then the corresponding getter method throws <code>DmtException</code> with the error code </code>FEATURE_NOT_SUPPORTED</code>. In case the Dmt Admin receives a <code>null</code> from a plugin (or other value it can not interpret as a proper result for a property getter method) it must also throw the appropriate <code>DmtException</code>.

Invalid tag on top level code (x=75,y=3) [p.253]

This interface collects basic DMT operations. The application programmers use these methods when they are interacting with a <code>DmtSession</code> which inherits from this interface. The <code>DmtDataPlugin</code> and the </code>DmtReadOnlyDataPlugin</code> interfaces also extend this interface. <p> If the admin or a data plugin does not support an optional DMT property (like <code>timestamp</code>) then the corresponding getter method throws <code>DmtException</code> with the error code </code>FEATURE_NOT_SUPPORTED</code>. In case the Dmt Admin receives a <code>null</code> from a plugin (or other value it can not interpret as a proper result for a property getter method) it must also throw the appropriate <code>DmtException</code>.

Invalid tag on top level code (x=71,y=8) [p.253]

This interface collects basic DMT operations. The application programmers use these methods when they are interacting with a <code>DmtSession</code> which inherits from this interface. The <code>DmtDataPlugin</code> and the </code>DmtReadOnlyDataPlugin</code> interfaces also extend this interface. <p> If the admin or a data plugin does not support an optional DMT property (like <code>timestamp</code>) then the corresponding getter method throws <code>DmtException</code> with the error code </code>FEATURE_NOT_SUPPORTED</code>. In case the Dmt Admin receives a <code>null</code> from a plugin (or other value it can not interpret as a proper result for a property getter method) it must also throw the appropriate <code>DmtException</code>.

Unexpected tag, assume para (x=25,y=5) [p.238]

The DmtAdmin interface is used to create DmtSession objects. The implementation of DmtAdmin should register itself in the OSGi service registry as a service. DmtAdmin is the entry point for applications to use the Dmt API. The ‹code›getSession‹/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(DmtAdmin.class.getName()); DmtAdmin factory = (DmtAdmin) context.getService(serviceRef); DmtSession session = factory.getSession("./OSGi/cfg"); session.createInteriorNode("./OSGi/cfg/mycfg"); ‹/pre›‹/blockquote›

Invalid tag on top level blockquote (x=39,y=10) [p.238]

The DmtAdmin interface is used to create DmtSession objects. The implementation of DmtAdmin should register itself in the OSGi service registry as a service. DmtAdmin is the entry point for applications to use the Dmt API. The ‹code›getSession‹/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(DmtAdmin.class.getName()); DmtAdmin factory = (DmtAdmin) context.getService(serviceRef); DmtSession session = factory.getSession("./OSGi/cfg"); 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; version=1.0
```

11.12.1 Summary

- Dmt - A collection of DMT manipulation methods. [p.232]
- DmtAcl - The DmtAcl class represents structured access to DMT ACLs. [p.236]
- DmtAdmin - The DmtAdmin interface is used to create DmtSession objects. [p.238]
- DmtAlertItem - Data structure carried in an alert (client initiated notification). [p.239]
- DmtAlertSender - The Dmt Admin provides the DmtAlertSender service which can be used by clients to send notifications to the server. [p.240]
- DmtData - A data structure representing a leaf node. [p.241]
- DmtDataPlugin - An implementation of this interface takes the responsibility over a modifiable subtree of the DMT. [p.244]
- DmtDataType - A collection of constants describing the possible formats of a DMT node. [p.244]
- DmtException - Checked exception received when a DMT operation fails. [p.245]
- DmtExecPlugin - An implementation of this interface takes the responsibility of handling EXEC requests in a subtree of the DMT. [p.249]
- 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.249]
- DmtPermission - DmtPermission controls access to management objects in the Device Management Tree (DMT). [p.252]

- DmtPrincipalPermission - Indicates the callers authority to create DMT sessions in the name of a remote management server. [p.253]
- DmtReadOnly - This interface collects basic DMT operations. [p.253]
- DmtReadOnlyDataPlugin - An implementation of this interface takes the responsibility over a non-modifiable subtree of the DMT. [p.256]
- DmtSession - DmtSession provides concurrent access to the DMT. [p.257]

11.12.2 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.

11.12.2.1 public void copy(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` – with the following possible error codes NODE_ALREADY_EXISTS if newNodeUri already exists NODE_NOT_FOUND if nodeUri does not exist URI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if either URI is not within the current session's subtree COMMAND_NOT_ALLOWED if any of the implied Get or Add commands are not allowed, or if nodeUri is an ancestor of newNodeUri METADATA_MISMATCHDATA_STORE_FAILUREFORMAT_NOT_SUPPORTEDTRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.2 public void createInteriorNode(String nodeUri) throws DmtException

nodeUri The URI of the node

□ Create an interior node

Throws `DmtException` – with the following possible error codes URI_TOO_LONGINVALID_URIPERMISSION_DENIEDNODE_ALREADY_EXISTSOTHER_ERROR if the URI is not within the current session's subtree METADATA_MISMATCHDATA_STORE_FAILURETRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.3 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` – with the following possible error codes URI_TOO_LONGINVALID_URIPERMISSION_DENIEDNODE_ALREADY_EXISTSOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILED if the type string is invalid METADATA_MISMATCHDATA_STORE_FAILURETRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.4 public void createLeafNode(String nodeUri) throws DmtException

nodeUri The URI of the node

- Create a leaf node with default value. If a node does not have a default value defined by it's meta data, this method will throw a DmtException with error code METADATA_MISMATCH. The MIME type of the default node should also be specified by the meta data.

Throws `DmtException` – with the following possible error codes NODE_ALREADY_EXISTSURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWEDMETADATA_MISMATCHDATA_STORE_FAILURETRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.5 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, can not be null

- Create a leaf node with a given value. The node's MIME type is not explicitely specified, it will be derived from the meta data associated with this node. The meta data defining the possible type (if any) should allow only one MIME type, otherwise this method will fail with METADATA_MISMATCH. Nodes of null format can be created by using DmtData.NULL_VALUE as second argument.

Throws `DmtException` – with the following possible error codes NODE_ALREADY_EXISTSURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILED if the data is null COMMAND_NOT_ALLOWEDMETADATA_MISMATCHDATA_STORE_FAILUREFORMAT_NOT_SUPPORTEDTRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

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

nodeUri The URI of the node

value The value to be given to the new node, can not be null

mimeType The MIME type to be given to the new node. It can be null.

□ Create a leaf node with a given value and MIME type.

Throws `DmtException` – with the following possible error codes NODE_ALREADY_EXISTSURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWEDCOMMAND_FAILED if the data is null METADATA_MISMATCHDATA_STORE_FAILUREFORMAT_NOT_SUPPORTEDTRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.7 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWED if the node is permanent or non-deletable METADATA_MISMATCHDATA_STORE_FAILURETRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.8 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILED if the newName string is invalid COMMAND_NOT_ALLOWEDMETADATA_MISMATCHDATA_STORE_FAILURETRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.9 public void rollback( ) throws DmtException

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

Throws `DmtException` – with the following possible error codes ROLLBACK_FAILED in case the rollback did not succeed

FEATURE_NOT_SUPPORTED in case the session was not created using the LOCK_TYPE_ATOMIC lock type.

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.10 public void setDefaultNodeValue(String nodeUri) throws DmtException

nodeUri The URI of the node

- Set the value of a leaf node to it's default as defined by the node's meta data. The method throws exception if there is no default defined.

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDCOMMAND_FAILEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWED if the specified node is not a leaf node METADATA_MISMATCHDATA_STORE_FAILURETRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.11 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDFEATURE_NOT_SUPPORTEDCOMMAND_FAILED if the title string is too long or contains not allowed characters OTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWEDMETADATA_MISMATCHDATA_STORE_FAILURETRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

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

nodeUri The URI of the node

type The type of the node

- Set 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree FEATURE_NOT_SUPPORTEDCOMMAND_FAILED if the type string is null or invalid COMMAND_NOT_ALLOWEDMETADATA_MISMATCHDATA_STORE_FAILUREFORMAT_NOT_SUPPORTEDTRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.2.13 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. Nodes of null format can be set by using DmtData.NULL_VALUE as second argument.

- Set the value of a leaf node.

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDCOMMAND_FAILED if the data is null OTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWED if the specified node is not a leaf node METADATA_MISMATCHDATA_STORE_FAILUREFORMAT_NOT_SUPPORTEDTRANSACTION_ERROR

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.3 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 DmtAdmin.getTree), as well as "*" indicating any principal.

11.12.3.1 public static final int ADD = 2

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

11.12.3.2 public static final int DELETE = 8

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

11.12.3.3 public static final int EXEC = 16

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

11.12.3.4 public static final int GET = 1

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

11.12.3.5 public static final int REPLACE = 4

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

11.12.3.6 public DmtAcl( )

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

11.12.3.7 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

11.12.3.8 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

11.12.3.9 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.

11.12.3.10 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.

11.12.3.11 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.

11.12.3.12 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.

11.12.3.13 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

11.12.3.14 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.

11.12.3.15 public synchronized String toString( )

- Give the canonic string representation of this ACL.

Returns The string representation as defined in OMA DM.

11.12.4 public interface DmtAdmin

The DmtAdmin interface is used to create DmtSession objects. The implementation of DmtAdmin should register itself in the OSGi service registry as a service. DmtAdmin is the entry point for applications to use the Dmt API. The getSession methods are used to open a session on a specified subtree of the DMT. A typical way of usage:

```
    serviceRef = context.getServiceReference(DmtAd-
min.class.getName());
    DmtAdmin factory = (DmtAdmin) context.getService(service-
Ref);
    DmtSession session = factory.getSession(”./OSGi/cfg”);
    session.createInteriorNode(”./OSGi/cfg/mycfg”);
```

11.12.4.1 public DmtSession getSession(String subtreeUri) throws DmtException

subtreeUri The subtree on which DMT manipulations can be performed within the returned session. If you want to use the whole DMT then use “.” as subtree URI.

- Opens a DmtSession for local usage on a given subtree of the DMT with non transactional write lock. This call is equivalent to the following: getSession(null, subtreeUri, DmtSession.LOCK_TYPE_EXCLUSIVE)

Returns a DmtSession object on which DMT manipulations can be performed

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URITIMEOUT

11.12.4.2 public DmtSession getSession(String subtreeUri, int lockMode) throws DmtException

subtreeUri The subtree on which DMT manipulations can be performed within the returned session. If you want to use the whole DMT then use “.” as subtree URI.

lockMode One of the locking modes specified in DmtSession

- Opens a DmtSession for local usage on a specific DMT subtree with a given locking mode. This call is equivalent to the following: getSession(null, subtreeUri, lockMode)

Returns a DmtSession object on which DMT manipulations can be performed

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIOTHER_ERROR if the lockMode is unknown TIMEOUT

11.12.4.3 public DmtSession getSession(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. If you want to use the whole DMT then use "." as subtree URI.

lockMode One of the locking modes specified in DmtSession

- Opens a DmtSession on a specific DMT subtree using a specific locking mode on behalf of a remote principal. If local management applications are using this method then they should provide null as the first parameter. Alternatively they can use other forms of this method without providing a principal string. This method is guarded by DmtPrincipalPermission.

Returns a DmtSession object on which DMT manipulations can be performed

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIOTHER_ERROR if the lockMode is unknown TIMEOUT

`SecurityException` – if the caller does not have the required DmtPrincipalPermission

11.12.5 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.

11.12.5.1 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

11.12.5.2 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.

11.12.5.3 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.

11.12.5.4 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.

11.12.5.5 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.

11.12.5.6 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.

11.12.5.7 public String toString( )

11.12.6 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.

11.12.6.1 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. Can be null if not needed.

- 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` – with the following possible error codes
ALERT_NOT_ROUTED when the alert can not be routed to the server
FEATURE_NOT_SUPPORTED in case of locally initiated sessions
REMOTE_ERROR in case of communication problems between the device and the server

11.12.6.2 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. Can be null if not needed.

- 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` – with the following possible error codes
ALERT_NOT_ROUTED when the alert can not be routed to the server
FEATURE_NOT_SUPPORTED in case of locally initiated sessions
REMOTE_ERROR in case of communication problems between the device and the server

11.12.6.3 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. Can be null if not needed.

- 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` – with the following possible error codes
ALERT_NOT_ROUTED when the alert can not be routed to the server
FEATURE_NOT_SUPPORTED in case of locally initiated sessions
REMOTE_ERROR in case of communication problems between the device and the server

11.12.7 public class DmtData

A data structure representing a leaf node. This structure represents only the value and the format property of the node, all other properties of the node (like MIME type) can be set and read using the Dmt and DmtReadOnly interfaces.

Different constructors are available to create nodes with different formats. Nodes of null format can be created using the static DmtData.NULL_VALUE constant instance of this class.

11.12.7.1 public static final int FORMAT_BINARY = 8

The node holds an OMA DM binary value. The value of the node corresponds to the Java byte[] type.

11.12.7.2 public static final int FORMAT_BOOLEAN = 4

The node holds an OMA DM bool value.

11.12.7.3 public static final int FORMAT_INTEGER = 1

The node holds an integer value. Note that this does not correspond to the Java int type, OMA DM integers are unsigned.

11.12.7.4 public static final int FORMAT_NODE = 64

Format specifier of an internal node. A DmtData instance can not have this value. This is used only as a return value of the DmtMetaNode.getFormat() method.

11.12.7.5 public static final int FORMAT_NULL = 32

The node holds an OMA DM null value. This corresponds to the Java null type.

11.12.7.6 public static final int FORMAT_STRING = 2

The node holds an OMA DM chr value.

11.12.7.7 public static final int FORMAT_XML = 16

The node holds an OMA DM xml value.

11.12.7.8 public static DmtData NULL_VALUE

Constant instance representing a leaf node of null format.

11.12.7.9 public DmtData(String str)

str The string value to set

□ Create a DmtData instance of chr format with the given string value.

11.12.7.10 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.

11.12.7.11 public DmtData(int integer)

integer The integer value to set

- Create a DmtData instance of int format and set its value.

11.12.7.12 public DmtData(boolean bool)

bool The boolean value to set

- Create a DmtData instance of bool format and set its value.

11.12.7.13 public DmtData(byte[] bytes)

bytes The byte array to set

- Create a DmtData instance of bin format and set its value.

11.12.7.14 public boolean equals(Object obj)

11.12.7.15 public byte[] getBinary( ) throws DmtException

- Gets the value of a node with binary format

Returns The binary value

Throws `DmtException` – with the error code OTHER_ERROR if the format of the node is not binary

11.12.7.16 public boolean getBoolean( ) throws DmtException

- Gets the value of a node with boolean format

Returns The boolean value

Throws `DmtException` – with the error code OTHER_ERROR if the format of the node is not boolean

11.12.7.17 public int getFormat( )

- Get the node's format, expressed in terms of type constants defined in this class. 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.

11.12.7.18 public int getInt( ) throws DmtException

- Gets the value of a node with integer format

Returns The integer value

Throws `DmtException` – with the error code OTHER_ERROR if the format of the node is not integer

11.12.7.19 public String getString( ) throws DmtException

- Gets the value of a node with chr format

Returns The string value

Throws `DmtException` – with the error code OTHER_ERROR if the format of the node is not chr

11.12.7.20 public String getXml( ) throws DmtException

- Gets the value of a node with xml format

Returns The xml value

Throws `DmtException` – with the error code OTHER_ERROR if the format of the node is not xml

11.12.7.21 public int hashCode( )

11.12.7.22 public String toString( )

- 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

11.12.8 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 instead.

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.

11.12.8.1 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` –

11.12.8.2 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

11.12.9 public interface DmtDataType

A collection of constants describing the possible formats of a DMT node.

11.12.9.1 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.

11.12.9.2 public static final int BOOLEAN = 3

The node holds an OMA DM bool value.

11.12.9.3 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.

11.12.9.4 public static final int NODE = 7

The node is an internal node.

11.12.9.5 public static final int NULL = 6

The node holds an OMA DM null value. This corresponds to the Java null type.

11.12.9.6 public static final int STRING = 2

The node holds an OMA DM chr value.

11.12.9.7 public static final int UNSPECIFIED = 8

The format of the node is not specified by meta data. It can take one of several formats run time.

11.12.9.8 public static final int XML = 5

The node holds an OMA DM xml value.

11.12.10 public class DmtException extends Exception

Checked exception received when a DMT operation fails. Beside the exception message, a DmtException always contains an error code (one of the constants specified in this class), and may optionally contain the URI of the related node, and information about the cause of the exception.

The cause (if specified) can either be a single Throwable instance, or a list of such instances if several problems occured during the execution of a method. An example for the latter is the close method of DmtSession that tries to close multiple plugins, and has to report the exceptions of all failures.

Getter methods are provided to retrieve the values of the additional paramters, and the printStackTrace methods are extended to print the stack trace of all causing throwables as well.

11.12.10.1 public static final int ALERT_NOT_ROUTED = 5

An alert can not be sent from the device to the Remote Management Server because of missing routing information. This error code does not correspond to any OMA DM response status code.

11.12.10.2 **public static final int COMMAND_FAILED = 500**

The recipient encountered an unexpected condition which prevented it from fulfilling the request. This error code corresponds to the OMA DM response status code 500.

11.12.10.3 **public static final int COMMAND_NOT_ALLOWED = 405**

The requested command is not allowed on the target node. Examples are executing a non-executable node or trying to read the value of an interior node. This error code corresponds to the OMA DM response status code 405.

11.12.10.4 **public static final int CONCURRENT_ACCESS = 4**

An error occurred related to concurrent access of nodes. For example a configuration node was deleted through the Config Admin while the node was manipulated via DMT. This error code does not correspond to any OMA DM response status code.

11.12.10.5 **public static final int DATA_STORE_FAILURE = 510**

An error related to the recipient data store occurred while processing the request. This error code corresponds to the OMA DM response status code 510.

11.12.10.6 **public static final int FEATURE_NOT_SUPPORTED = 406**

The requested command failed because an optional feature in the request was not supported. This error code corresponds to the OMA DM response status code 406.

11.12.10.7 **public static final int FORMAT_NOT_SUPPORTED = 415**

Unsupported media type or format. This error code corresponds to the OMA DM response status code 415.

11.12.10.8 **public static final int INVALID_URI = 3**

The received URI string was null, contained not allowed characters or was not parseable to a valid URI. This error code does not correspond to any OMA DM response status code.

11.12.10.9 **public static final int METADATA_MISMATCH = 2**

Invalid data, operation failed bacause of meta data restrictions. Examples can be violating referential integrity constraints or exceeding maximum node value limits, etc. This error code does not correspond to any OMA DM response status code.

11.12.10.10 **public static final int NODE_ALREADY_EXISTS = 418**

The requested Add or Copy command failed because the target already exists. This error code corresponds to the OMA DM response status code 418.

11.12.10.11 **public static final int NODE_NOT_FOUND = 404**

The requested target node was not found. No indication is given as to whether this is a temporary or permanent condition. This error code corresponds to the OMA DM response status code 404.

11.12.10.12 **public static final int OTHER_ERROR = 0**

An error occurred that does not fit naturally into any of the other error categories. This error code does not correspond to any OMA DM response status code.

11.12.10.13 **public static final int PERMISSION_DENIED = 425**

The requested command failed because the sender does not have adequate access control permissions (ACL) on the target. This error code corresponds to the OMA DM response status code 425.

11.12.10.14 **public static final int REMOTE_ERROR = 1**

A device initiated remote operation failed. This error code does not correspond to any OMA DM response status code.

11.12.10.15 **public static final int ROLLBACK_FAILED = 516**

The rollback command was not completed successfully. It should be tried to recover the client back into original state. This error code corresponds to the OMA DM response status code 516.

11.12.10.16 **public static final int TIMEOUT = 7**

Creation of a session timed out because of another ongoing session. This error code does not correspond to any OMA DM response status code. OMA has several status codes related to timeout, but these are meant to be used when a request times out, not when a session can not be established.

11.12.10.17 **public static final int TRANSACTION_ERROR = 6**

This error is caused by one of the following situations: An updating method within an atomic session can not be executed because the underlying plugin does not support atomic transactions. //todo This error code does not correspond to any OMA DM response status code.

11.12.10.18 **public static final int URI_TOO_LONG = 414**

The requested command failed because the target URI is too long for what the recipient is able or willing to process. This error code corresponds to the OMA DM response status code 414.

11.12.10.19 **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.

11.12.10.20 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.

11.12.10.21 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.

11.12.10.22 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).

11.12.10.23 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.

11.12.10.24 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

11.12.10.25 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

11.12.10.26 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

11.12.10.27 public void printStackTrace(PrintStream s)

s PrintStream to use for output

- Prints the exception and its backtrace to the specified print stream. Any causes that were specified for this exception are also printed, together with their backtraces.

11.12.10.28 public void printStackTrace(PrintWriter s)

s PrintWriter to use for output

- Prints the exception and its backtrace to the specified print writer. Any causes that were specified for this exception are also printed, together with their backtraces.

11.12.11 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.

11.12.11.1 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). Can be null.

- 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` –

11.12.12 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 and regular expressions.

11.12.12.1 public static final int CMD_ADD = 0

11.12.12.2 public static final int CMD_DELETE = 1

11.12.12.3 public static final int CMD_EXECUTE = 2

11.12.12.4 public static final int CMD_GET = 4

11.12.12.5 public static final int CMD_REPLACE = 3

11.12.12.6 public static final int DYNAMIC = 1

11.12.12.7 public static final int PERMANENT = 0

11.12.12.8 public boolean can(int operation)

operation One of the DmtMetaNode.CMD_... constants.

- Check whether the given operation is valid for this node.

Returns false if the operation is not valid for this node or the operation code is not one of the allowed constants.

11.12.12.9 public DmtData getDefault( )

- Get the default value of this node if any.

Returns The default value or null if not defined.

11.12.12.10 public String getDescription( )

- Get the explanation string associated with this node

Returns Node description string

11.12.12.11 public int getFormat( )

- Get the node's format, expressed in terms of type constants defined in DmtData. If there are multiple formats allowed for the node then the format constants ar OR-ed. 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.

11.12.12.12 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.

11.12.12.13 public int getMaxOccurrence( )

- Get the number of maximum occurrences of this type of nodes on the same level in the DMT. Returns Integer.MAX_VALUE if there is no upper limit. Note that if the occurrence is greater than 1 then this node can not have siblings with different metadata. That is, if different type of nodes coexist on the same level, their occurrence can not be greater than 1.

Returns The maximum allowed occurrence of this node type

11.12.12.14 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. If there is a default value defined for this node then the associated MIME type (if any) must be the first element of the list.

11.12.12.15 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.

11.12.12.16 public String getNameRegExp( )

- Get the regular expression associated with the name of this node if any.

Returns The regular expression associated with the name of this node or null if not defined.

11.12.12.17 public String getRegExp( )

- Get the regular expression associated with the value of 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.

11.12.12.18 public int getScope( )

- Return the scope of the node. Valid values are DmtMetaNode.PERMANENT and DmtMetaNode.DYNAMIC. 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 DmtMetaNode.PERMANENT or DmtMetaNode.DYNAMIC

11.12.12.19 public String[] getValidNames( )

- Return an array of Strings if valid names are defined for the node, or null otherwise

Returns the valid values for this node name, or null if not defined

11.12.12.20 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

11.12.12.21 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

11.12.12.22 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

11.12.12.23 public boolean isLeaf( )

- Check whether the node is a leaf node or an internal one

Returns true if the node is a leaf node

11.12.12.24 public boolean isZeroOccurrenceAllowed( )

- Check whether zero occurrence of this node is valid

Returns true if zero occurrence of this node is valid

11.12.13 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.

11.12.13.1 public DmtPermission(String dmturi, String actions)

dmturi URI of the management object (or subtree).

actions OMA DM actions allowed.

- Creates a new DmtPermission object for the specified DMT URI with the specified actions.

11.12.13.2 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.

11.12.13.3 public String getActions( )

- Returns the String representation of the action list.

Returns Action list for this permission object.

11.12.13.4 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.

11.12.13.5 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.

11.12.13.6 public PermissionCollection newPermissionCollection( )

- Returns a new PermissionCollection object for storing DMTPermission objects.

Returns the new PermissionCollection.

11.12.14 public class DmtPrincipalPermission extends BasicPermission

Indicates the callers authority to create DMT sessions in the name of a remote management server. Only protocol adapters communicating with management servers should be granted this permission.

DmtPrincipalPermission has a target string which controls the name of the principal on whose behalf the protocol adapter can act. A wildcard is allowed in the target as defined by BasicPermission, for example a "*" means the adapter can create a session in the name of any principal.

11.12.14.1 public DmtPrincipalPermission(String target)

target Name of the principal

- Creates a new DmtPrincipalPermission object with its name set to the target string

11.12.14.2 public DmtPrincipalPermission(String target, String actions)

target Name of the principal

actions ignored

- Creates a new DmtPrincipalPermission object using the 'canonic' two argument constructor.

11.12.15 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. The DmtDataPlugin and the DmtReadOnlyDataPlugin interfaces also extend this interface.

If the admin or a data plugin does not support an optional DMT property (like timestamp) then the corresponding getter method throws DmtException with the error code

FEATURE_NOT_SUPPORTED . In case the Dmt Admin receives a null from a plugin (or other value it can not interpret as a proper result for a property getter method) it must also throw the appropriate DmtException.

11.12.15.1 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.

Throws `DmtException` – with the following possible error codes COMMAND_FAILED if an underlying plugin failed to close DATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.2 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. The returned array must not contain null entries. If a plugin returns null as an array element, then the admin must remove it from the array.

Returns The list of children node names as a string array or null if the node has no children.

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILEDCOMMAND_NOT_ALLOWEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.3 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.

  This method is inherited in the DmtSession and also in the Data Plugin interfaces, but the semantics of the method is slightly different in these two cases. Meta data can be supported at the engine level (i.e. in Dmt Admin), and its support by plugins is an optional (and advanced) feature. It can be used, for example, when a data plugin is implemented on top of a data store or another API that has their own metadata, such as a relational database, in order to avoid metadata duplication and inconsistency. The meta data specific to the plugin returned by the plugin's getMetaNode() method is complemented by the engine level meta data. The DmtMetaNode the client receives on the DmtSession.getMetaNode() call is the combination of the meta data returned by the data plugin plus the meta data returned by the Dmt Admin. If there are differences in the meta data elements known by the plugin and the Dmt Admin then the plugin specific elements take precedence.

Returns a DmtMetaNode which describes meta data information

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILEDCOMMAND_NOT_ALLOWEDFEATURE_NOT_SUPPORTEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.4 public int getNodeSize(String nodeUri) throws DmtException

nodeUri The URI of the node

- Get the size of the data in the node in bytes. Throws DmtException with the error code COMMAND_NOT_ALLOWED if issued on an interior node.

Returns The size of the node

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILEDCOMMAND_NOT_ALLOWEDFEATURE_NOT_SUPPORTEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.5 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILEDCOMMAND_NOT_ALLOWEDFEATURE_NOT_SUPPORTEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.6 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILEDCOMMAND_NOT_ALLOWEDFEATURE_NOT_SUPPORTEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.7 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILEDCOMMAND_NOT_ALLOWEDFEATURE_NOT_SUPPORTEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.8 public DmtData getNodeValue(String nodeUri) throws DmtException

nodeUri The URI of the node to retrieve

- Get the data contained in a leaf node.

Returns The data of the leaf node

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILEDCOMMAND_NOT_ALLOWED if the specified node is not a leaf node DATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.9 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 a leaf node is changed. The version property is undefined for interior nodes.

Returns The version of the node

Throws `DmtException` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_FAILEDCOMMAND_NOT_ALLOWEDFEATURE_NOT_SUPPORTEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.15.10 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

Throws `IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.16 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.

11.12.16.1 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` –

11.12.17 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.

Each method of the DmtSession can throw InvalidStateException in case the session is invalidated because of timeout.

11.12.17.1 public static final int LOCK_TYPE_ATOMIC = 2

LOCK_TYPE_ATOMIC is an exclusive lock with transactional functionality. Commands of an atomic session will either fail or succeed together, if a single command fails then the whole session will be rolled back.

11.12.17.2 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.

11.12.17.3 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.

11.12.17.4 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. Can be null.

□ 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWED if the node is non-executable COMMAND_FAILED if no DmtExecPlugin is associated with the node DATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.17.5 public int getLockType( )

□ Gives the type of lock the session currently has.

Returns One of the LOCK_TYPE_... constants.

Throws `IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.17.6 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.17.7 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

Throws `IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.17.8 public String getRootUri( )

□ Get the root URI associated with this session. Gives "." if the session was created without specifying a root, which means that the target of this session is the whole DMT.

Returns the root URI

Throws `IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.17.9 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

Throws `IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.17.10 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWED

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.12.17.11 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` – with the following possible error codes NODE_NOT_FOUNDURI_TOO_LONGINVALID_URIPERMISSION_DENIEDOTHER_ERROR if the URI is not within the current session's subtree COMMAND_NOT_ALLOWEDDATA_STORE_FAILURE

`IllegalStateException` – if the session is invalidated because of timeout, or if the session is already closed or rolled back.

11.13 References

[17] *SyncML Device Management Tree and Description, Version 1.1.2*
http://www.openmobilealliance.org/release_program/dm_v112.html

[18] *IETF RFC2578. Structure of Management Information*
Version 2 (SMIv2), http://www.ietf.org/rfc/rfc2578.txt

[19] *Java™ Management Extensions Instrumentation and Agent Specification*
v1.2, October 2002, http://java.sun.com/products/JavaManagement/

[20] *JSR 9 - Federated Management Architecture (FMA) Specification*
Version 1.0, January 2000, http://www.jcp.org/en/jsr/detail?id=9

[21] *WBEM Profile Template, DSP1000*
Status: Draft, Version 1.0 Preliminary, March 11, 2004
http://www.dmtf.org/standards/wbem

[22] *SNMP*
http://www.wtcs.org/snmp4tpc/snmp_rfc.htm#rfc

[23] *RFC 2396 Uniform Resource Identifiers (URI): Generic Syntax*
http://www.ietf.org/rfc/rfc2396.txt

[24] *MIME Media Types*
http://www.iana.org/assignments/media-types/

12 Monitor Admin Service Specification

Version 1.0

12.1 Introduction

Applications and services may publish indicator variables (Key Performance Indicators, KPI) that management systems can query obtaining information about the behaviour of the application. For example, a bundle could publish KPIs for a number key VM variables like the amount of avaible memories.

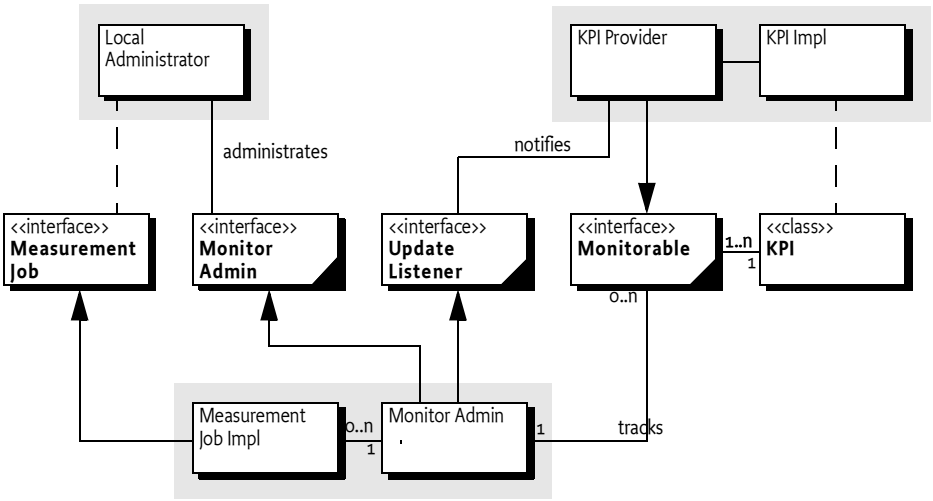
KPIs can be used in performance management, fault management as well as in customer relations management systems.

This specification outlines how a bundle can publish KPIs and how administrative bundles can find out about KPIs as well as read their values.

12.1.1 Entities

- *KPI* – Application specific variables that a *KPI Provider* publishes with a Monitorable service to the Monitor Admin. A KPI has a value, a description and a time stamp. KPI Values can numbers, strings or general objects.
- *KPI provider* – A bundle which has a number of KPIs that it publishes with a Monitorable service.
- *Monitor Admin* – Provides unified and secure access to available KPIs as well as providing a function create Measurement Jobs that periodically query the KPIs.
- *Monitorable* – A service that is registered by a KPI provider to publish its KPIs.
- *Measurement job* – An event or time based periodic query of a given set of KPIs. When a KPI value is updated within a Measurement Job, the Monitor Admin must generate an event.
- *Local Administrator* – A management application which uses the Monitor Admin service to query KPIs and to initiate Measurement Jobs.
- *KPI name* – The string name a KPI has within a KPI Provider.
- *KPI Path* – A string that uniquely identifies the KPI in an OSGi environment. It consists of the PID of the Monitorable service and the KPI name separated by a slash.

Figure 53 Monitor Admin Diagram org.osgi.service.monitor package



12.1.2 Operations

A bundle that provides a KPI must register a Monitorable service. This service is used by the Monitor Admin to get KPIs and provide meta information to clients.

Clients can use the Monitor Admin to obtain KPIs in a protected way. Clients can also create Monitoring Jobs. These Monitoring Jobs send out notifications to the clients when the value changes or periodically.

12.2 Key Performance Indicators

A KPI is a *Key Performance Indicator*. It is a simple value that is published from a KPI provider to the Monitor Admin. A KPI Provider can publish many KPIs at the same time by registering a Monitorable service. A KPI combines a name, a path, a value, a time stamp, a collection method, and a description.

I think the KPI class must be final and immutable? Once it is created it should not be changeable. Otherwise a KPI that is handed to multiple parties could be modified without the other guy knowing

The name KPI stinks :-)

The abbreviation is not used consistently through the method names. We should use getKPI to consistent with the rest of the spec.

12.2.1 KPI Name

Each KPI must have a unique identity in the scope of a KPI provider. This identity can be obtained with the getID() method. A KPI identity must fulfil the following requirements:

- It must not contain any reserved characters as defined in [27] *RFC-2396 Uniform Resource Identifiers (URI): Generic Syntax*
- The length must be limited to 20 characters max
- It must be unique in the scope of the KPI Provider.

The name should be descriptive and concise.

It would be nice if we could share a definition of the character set from the framework document.

12.2.2 KPI Path

KPIs are accessible through a Monitorable service. The KPI can therefore provide a path to itself that is sufficient to retrieve it again (or an update). This path is returned with the getPath() method. A path has the following syntax:

```
pki-path   ::= pid '/' name
pid        ::= unique-name
name       ::= [^ <reserved characters> ]+
```

12.2.3 Value

A KPI provides the type of its value with the getType()method. The return value of this method can take the following values:

- TYPE_INTEGER – A signed numeric value that fits in a Java int type. The associated method to retrieve the value is getInteger().

If this method is required, it should be called getInt to discriminate it from a method that returns an object.

- TYPE_FLOAT – A floating point value that fits in a Java float type. The associated method to retrieve the value is getFloat().
- TYPE_STRING – A String object. The associated method to retrieve the value is getString().
- TYPE_OBJECT – Any object. The associated method to retrieve the value is getObject().

This API looks overly complicated because it allows object, but makes a rather arbitrary selection of int, string and float. Suggest to just return an object type and the user figure it out with instanceof

If the a method is called that does not match the return value of the getType method, the KPI must throw an Illegal State Exception.

12.2.4 Time Stamp

The time stamp must reflect the time that the measurement was taken from the standard Java System.currentTimeMillis method. The time stamp can be obtained with the getTimeStamp() method.

12.2.5 Collection Method

This specification is compatible with terminology used in [26] *ETSI Performance Management [TS 132 403]*. An important concept of a KPI is the way it was collected, this is called the *collection method*. The collection method is independent of how (if and when) the reporting of the KPIs happens. The collection method is part of the KPI's definition and can not be changed. The collection method of a KPI can be obtained with the getCollectionMethod() method.

The ETSI document defines the following collection methods:

- CM_CC – A numeric counter whose value can only increase, except when the KPI is reset. An example of a CC is a variable which stores the number of incoming SMSs handled by the protocol driver since it was started.
- CM_GAUGE – A numeric counter whose value can vary up or down. An example of a GAUGE is a variable which stores the current battery level percentage. The value of the KPI must be the absolute value not a difference.
- CM_DER – (Discrete Event Registration) A status variable (numeric or string) which can change when a certain event happens in the system one or more times. The event which fires the change of the KPI is typically some event like the arrival of an SMS. The definition of a DER counter contains an integer N which means how many events it takes for the counter to change its value. The most usual value for N is 1, but if N is greater than 1 then it means that the variable changes after each Nth event.
- - CM_SI – (Status Inspect) The most general status variable which can be a string or numeric. An example of an SI is a string variable which contains the name of the currently logged in user.

If a KPI is of array or Vector type then the KPI's collection method is valid for *all* elements in the array or Vector object.

12.2.6 Description

A human readable description of this KPI.It is obtained with the getDescription() method.

How is localization required?

12.2.7 Resetting KPIs

In certain cases it is required to reset a KPI back to its initial value. This method is placed on the Monitorable interface because the KPI provides a single measurement. The resetKpi(String) method will reset counters. After the reset, the KPI Provider must call the Update Listener services update method to notify the change.

what are those cases? this seems to be very much counter intuitive? I.e. "monitoring" implies non destructable .. This gets closer and closer to Diagnostics.

Shouldn't the description not also be a method on Monitorable? It feels kind of expensive to keep it in the KPI.

The design of the KPI would have been better if there had been a stricter separation between the statics the dynamics.

```
public class Indicator {
        Indicator( IndicatorType type, Object value ) {...}
        long getTimeStamp() {...}
        Object getValue() {}
        IndicatorType getType() {..}
}

public class IndicatorType {
        String          description;
        String          name;
        String          dataType;
        boolean         notifiesOnChange;
        ../ access methods
}
```

12.3 KPI Provider

Applications that want to participate in the Monitoring Admin service are KPI Providers. A KPI Provider must register a Monitorable service. This service is tracked by the Monitor Admin service that will provide access to the KPIs to the local admins.

For example, the following code shows how a bundle could provide a KPI with the current amount of memory.

```
public class KPITest implements BundleActivator, Monitorable
{
   public void start(BundleContext context) {
      Hashtable ht = new Hashtable();
      ht.put("service.pid", "com.acme.foo");
      context.registerService(
         Monitorable.class.getName(), this, ht);
   }

   public void stop(BundleContext context) {}
```

```
    public String[] getKpiNames() {
       return new String[] {"memory.free"};
    }

    public KPI getKpi(String id) throws
          IllegalArgumentException {
       if ("memory.free".equals(id))
          return new KPI("com.acme.foo", "memory.free",
             "Free memory", KPI.CM_GAUGE,
             Runtime.getRuntime().freeMemory());
       else
          throw new IllegalArgumentException(
             "Invalid KPI name " + id);
    }

    public boolean notifiesOnChange(String id)
       throws IllegalArgumentException {
       return false;
    }

    public boolean resetKpi(String id)
       throws IllegalArgumentException {
       return false;
    }
    public String[] getKpiPaths() {
       // ### Silly to force the bundle to implement this
       return new String[] {"com.acme.foo/memory.free"};
    }

    public org.osgi.service.monitor.KPI[] getKpis() {
       // ### Same could be done by the Monitor Admin
       return new KPI[] { getKpi("memory.free") };
    }
 }
```

If you want to make this easy to implement for bundle programmers (and only if all bundle implement this will it be successful, than writing a Monitorable must be simpler.

12.3.1 Providing Notifications

If a Monitorable service returns true for the notifiesOnChange(String) method then it must notify all Update Listener services when the related KPI changes. These KPIs are called dynamic KPIs.

After a value related to a dynamic KPI is changed, the KPI Provider must get *all* Update Listener services and call its updated(KPI). Monitor Admin uses this notification to send out a generic event on the Event Channel. Usually there is only a single Update Listener service.

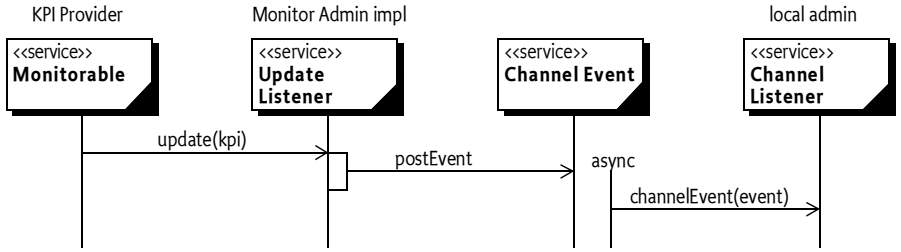
Why is the KPI given as a parameter? It is ignored in the event

Is there no way to turn the events on and off? Sounds awfully expensive?

The name of the Update Listener is rather bad because it does not show a link to MOnitor xxx. Maybe MonitorListener? Alternatively, it would probably even be better if the notification went through the Monitor Admin, why have the listener?

Figure 54 shows a sequence diagram for such an update.

Figure 54 Notification on Update



12.4 Using Monitor Admin Service

The Monitor Admin service provides unified access to the KPIs in the system. It provides security checking, resolution of the KPI paths and scheduling of periodic or event based Measurement Jobs.

12.4.1 KPI Administration

The Monitor Admin provides the following methods for accessing the KPIs:

- getKPI(String) – Return a KPI, given a KPI path. The KPI is immutable.
- getMonitorable(String) – Given a PID, return a Monitorable service object. The service must currently be registered. The objects are actual service objects owned by the Monitor Admin. Callers must not hold on to references to these objects.
- getMonitorables() – Return a list of all Monitorable services currently registered.The objects are actual service objects owned by the Monitor Admin. Callers must not hold on to references to these objects.

Why does anybody need the monitorables? Normal local admins should never need them? They are just a communication mechanism between the Monitor Admin and the KPI provider? This means the Monitorable can get into the wrong hands, which is a pity.There is also all kind of stale references problems. If there are specialized applications, they could get permission to use the Monitorable services from the registry

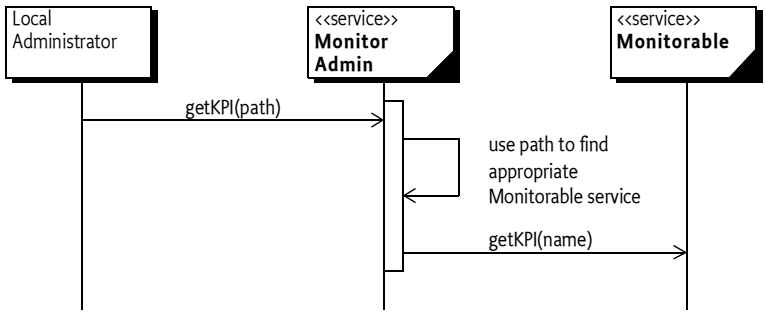
I also think that the Monitor Admin then should provide a reset method.

Figure 55 *KPI request* is the simple sequence diagram for getting a KPI from the Monitor Admin. The local administrator requests a KPI from the Monitor Admin service with the getKPI(String) method. Its sole method specifies a path to the KPI. The first part of the path is the PID of the Monitorable service, the second part is the name of the KPI. The parts are separated by a slash ('/' \u####). For example:

```
com.acme.foo/memory.free
```

The Monitor Admin service finds the associated Monitorable service by looking for a Monitorable service with the given PID (com.acme.foo). It will then query the Monitorable service for the KPI memory.free, which is then subsequently returned to the local administrator.

Figure 55 *KPI request*



12.4.2 Monitoring jobs

A local administrator can issue a monitoring job. A monitoring job consists of a set of KPIs and *reporting rules* for those KPIs. Reporting rules for KPIs are set in a monitoring job. The same KPI can participate in more then one monitoring job.

Monitoring Jobs are created with the startJob(String,String[],int,int) method on the Monitor Admin service. It takes the following parameters.

- *Initiator* – (String) This identity string is placed in the event as the listener.id property. This makes it straightforward to let the local administrator to receive only events that are addressed to it using the Event Channel service's filtering model.
- *KPIs* – A list of the KPI paths that must be monitored.

The schedule and count parameters have different meaning depending on periodic based notifications or event based notifications.

- *Periodic* – The data acquisition and reporting of measurements is periodic. For example, the current value of KPI myapp/number_of_threads must be reported with a period of one minute.
  - *Schedule* – Period in seconds, greater than 0, first sample is taken after the first period.
  - *Count* – Stop after this number of periods have run. This limits the number of acquisitions.
- *Event* – Changes in the KPI must be reported directly, or for every nth event. In this case it is required that the involved KPIs support the instant notification feature. Each Monitorable service can indicate which KPIs are dynamic with the notifiesOnChange(String) method that takes the KPI name as a parameter.
  - Schedule – Must be zero to activate Event based reporting.
  - Count – The number of measurements to take or zero for infinite. This number must be greater or equal to zero.

Is there an error when the event based method is used for a KPI that does not support modifications?

The schedule and count interaction is too complex imho

For example, a local administrator sets up a time based monitoring job to receive the com.acme.foo/RAB.FailEstabPSQueuing KPI updates periodically every minute with

```
startJob(
   "foo.bar",        // listener.id
    new String[] {"com.acme.foo/memory.free"},    // KPI paths
   15,               // seconds
    100          // 100 periods = 1500 seconds
);
```

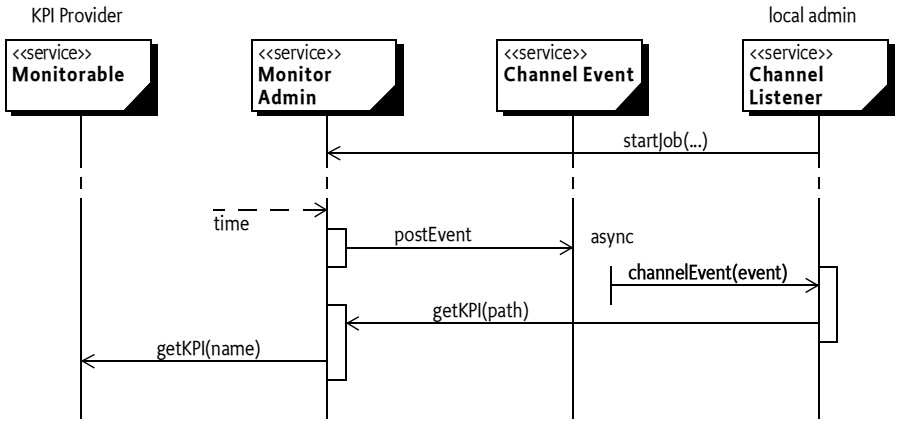
The local administrator should registers itself as an Channel Listener service with a filter on the listener.id event property that it matches the foo.bar id.

Subsequently, the Monitor Admin queries the com.acme.foo/memory.free KPI every 15 seconds. At each acquisition, the Monitor Admin sends a org.osgi.service.monitor.MonitorEvent event to the Event Channel. The event properties contain the listener.id foo.bar. The Event Channel service therefore updates the Channel Listener that was registered by the local admin. After receiving the events, the local admin can query the updated value of the KPI from the Monitor Admin service.

The events are repeated once every 15 seconds until either the administrator explicitly stops the job or when the Monitor Admin has sent 100 notifications.

With this description, the monitoring job is just some kind of cron task...The only thing the event does is wake up the local admin.

Figure 56 Time based monitoring job



12.5 Monitoring events

The Monitor Admin must send an event to the Event Manager whenever a KPI is updated within a locally initiated Monitoring Job. This can happen when:

- A KPI Provider reported the change through the Update Listener service.
- The KPI is queried from within a Monitoring Job by the Monitor Admin

Is this ALWYAYS the cases? Is the job not kind of arbitrary?

- The KPI was reset.

The topic of the event must be:

```
org.osgi.service.monitor.MonitorEvent
```

I think org.osgi.service.monitor should also work? Smaller?

The properties of the event are:

- monitorable.pid – (String) The PID of a Monitorable service that is related to the updated a KPI.

WHy is this not the KPI path?

- kpi.name – (String) The name of the updated KPI.

Related strings should have some association... monitorable.kpi? Same for other variables.

Is this going to be delivered asynchronous or synchronous?

- listener.id – (String or String[ ]) Name(s) representing the initiators of the monitoring job in which the KPI was updated. Listeners can use this field for filtering, so that they receive only events related to their own jobs. This property is only present when the event was generated by a monitoring job; it will not be provided in the event for events that are generated by notifications.

The updated value of the KPI is not present in the event, event listeners must query the value from the Monitor Admin service with the given monitorable.pid and kpi.name properties.

The event which fires a CM_KPI of CM_DER type is not related to the monitoring events.

Have no idea what this means?

12.6 DMT Access

Monitoring jobs can be started also remotely by a management server through Device Management Tree operations. Because of limitations in the DMT structure, in the remote case the job can contain only one KPI. The monitoring job has a method which tells whether it was started locally or remotely.

We have to discuss how to handle the DMT parts. I would like to discuss those in a separate document.

12.7 Security

Registering Monitorable services, querying KPIs and starting measurement jobs requires a Kpi Permission. If the entity issuing the operation does not have this permission, a Security Exception is thrown.

The KpiPermission uses a KPI path as target and recognizes the following actions:

- READ – Permit read access to KPIs. This permission is needed by a local admin to see a KPI. Monitor Admin service must filter all KPIs that a local has no permission for.
- RESET – Protects the Monitorable resetKpi method.
- PUBLISH – A KPI Provider must have this permission to publish a KPI. This does not forbid the KPI Provider to register the Monitorable. However, the Monitor Admin must not show a KPI to any local admin when the KPI Provider has no permission to publish that KPI.
- STARTJOB – Permission required by a local admin that wants to start a Monitoring Job.

The permissions are all checked by the Monitor Admin.

Further, the different actors must have the permissions as specified in Table 13 to operate correctly.

ServicePermission	KPI Provider	Local Admin	Monitor Admin
MonitorAdmin	-	GET	REGISTER
UpdateListener	GET	-	REGISTER
Monitorable	REGISTER	-	GET

Table 13 *Permission for the different actors*

There are currently no security exceptions indicated in the javadoc ..

It look like you need to have an DISCOVER actions as well that protects the methods that show what KPIs exist.

Need constants for the actions

The reset action can only be tested in the Monitorable implementation, not very secure because that is app code.

12.8 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; version=1.0
```

12.8.1 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.272]
- KpiPermission - Indicates the callers authority to publish, read or reset KPIs or to start monitoring jobs. [p.275]
- Monitorable - A Monitorable can provide information about itself in the form of KPIs. [p.276]
- MonitorAdmin - A MonitorAdmin implementation handles KPI query requests and measurement job control requests. [p.279]
- MonitoringJob - A Monitoring Job is a scheduled update of a set of KPIs. [p.280]
- UpdateListener - The UpdateListener is used by Monitorable services to send notifications when a KPI value is changed. [p.281]

12.8.2 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.

12.8.2.1 public static final int CM_CC = 0

Collection method type identifying 'Cumulative Counter' data collection.

12.8.2.2 public static final int CM_DER = 1

Collection method type identifying 'Discrete Event Registration' data collection.

12.8.2.3 public static final int CM_GAUGE = 2

Collection method type identifying 'Gauge' data collection.

12.8.2.4 public static final int CM_SI = 3

Collection method type identifying 'Status Inspection' data collection.

12.8.2.5 public static final int TYPE_FLOAT = 1

KPI type identifying float data.

12.8.2.6 public static final int TYPE_INTEGER = 0

KPI type identifying integer data.

12.8.2.7 public static final int TYPE_OBJECT = 3

KPI type identifying Object data.

12.8.2.8 public static final int TYPE_STRING = 2

KPI type identifying string data.

12.8.2.9 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

12.8.2.10 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

12.8.2.11 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

12.8.2.12 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

- Constructor for a KPI of Object type. The type of the object should be supported by the Configuration Admin. In case of arrays and vectors all elements of the collection must be of the same dynamic type. For arrays, this must also be the same as the component type of the array itself. E.g. an array of Objects is not allowed (even if it only contains Integer instances), but an array of Integers is.

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

12.8.2.13 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

12.8.2.14 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

12.8.2.15 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

12.8.2.16 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

12.8.2.17 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

12.8.2.18 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

12.8.2.19 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

12.8.2.20 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

12.8.2.21 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

12.8.2.22 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

12.8.2.23 public int getType( )

- Returns information on the data type of this KPI.

Returns one value of the set of type constants

12.8.2.24 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.

12.8.2.25 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

12.8.3 public class KpiPermission extends Permission

Indicates the callers authority to publish, read or reset KPIs or to start monitoring jobs. The target of the permission is the identifier of the KPI, the action can be read, publish, reset, startjob or the combination of these separated by commas.

12.8.3.1 public KpiPermission(String kpi, String actions)

kpi The identifier of the KPI in [Monitorable_id]/[KPI_id] format. The wildcard * is allowed in both fragments of the target string, but only at the end of the fragments. The following targets are valid: com.mycomp.myapp/queue_length, com.mycomp.*/queue*, com.mycomp.myapp/*, */*. The following targets are invalid: *.myapp/queue_length, com.*.myapp/*, *.

actions The allowed action(s): read, publish, startjob, reset, or the combination of these separated by commas. In case of the startjob action a minimal sampling interval can be optionally defined in the following form: startjob:n. Here n is the allowed minimal value of the schedule parameter of time based monitoring jobs the holder of this permission is allowed to initiate. If n is not specified or 0 then the holder of this permission is allowed to start monitoring jobs specifying any frequency.

- Create a KpiPermission object, specifying the target and actions.

12.8.3.2 public boolean equals(Object o)

o the object being compared for equality with this object

- Determines the equality of two KpiPermission objects. Two KpiPermission objects are equal if their target and action strings are equal.

Returns true if the two permissions are equal

12.8.3.3 public String getActions( )

- Get the action string associated with this permission

Returns the allowed actions separated by commas

12.8.3.4 public int hashCode( )

- Create an integer hash of the object. The hash codes of KpiPermissions p1 and p2 are the same if and only if p1.equals(p2)

Returns the hash of the object

12.8.3.5 public boolean implies(Permission p)

p the permission to be checked

- Determines if the specified permission is implied by this permission.

  This method returns false iff any of the following conditions are fulfilled for the specified permission:

  it is not a KpiPermission it has a broader set of actions allowed than this one it allows initiating time based monitoring jobs with a lower minimal sampling interval the target set of Monitorables is not the same nor a subset of the target set of Monitorables of this permission the target set of KPIs is not the same nor a subset of the target set of KPIs of this permission

Returns true if the given permission is implied by this permission

12.8.4 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). Also the length of the PID should be kept as small as possible. The PID will be used as a node name in the DMT and certain DMT implementations may have limits on node name length. The length limit is not specified in any standard, it is recommended not to use names longer than 20 characters.

A Monitorable may optionally support sending notifications when the status of its KPIs change.

Publishing KPIs requires the presence of the KpiPermission with the publish action string. This permission, however, is not checked during registration of the Monitorable service. Instead, the MonitorAdmin implemenatation must make sure that when a KPI is queried, it is shown only if the Monitorable is authorized to publish the given KPI.

12.8.4.1 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.

The entity which queries the KPI needs to hold KpiPermission with the read action present. The target field of the permission must match the KPI to be read.

Returns the KPI object

Throws `IllegalArgumentException` – if the path is invalid or points to a non existing KPI

`SecurityException` – if the caller does not hold KpiPermission with the read action or if the specified KPI is not allowed to be read as per the target field of the permission

12.8.4.2 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

12.8.4.3 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 MonitoringJob'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

12.8.4.4 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.

  The entity which queries the KPI list needs to hold KpiPermission with the read action present. The target field of the permission must match all the KPIs published by this Monitorable.

Returns the KPI objects published by this Monitorable instance

Throws `SecurityException` – if the caller does not hold KpiPermission with the read action or if there is any KPI published by the Monitorable which is not allowed to be read as per the target field of the permission

12.8.4.5 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

12.8.4.6 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.

  The entity that wants to reset the KPI needs to hold KpiPermission with the reset action present. The target field of the permission must match the KPI to be reset.

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

`SecurityException` – if the caller does not hold KpiPermission with the reset action or if the specified KPI is not allowed to be reset as per the target field of the permission

12.8.5 public interface MonitorAdmin

A MonitorAdmin implementation handles KPI query requests and measurement job control requests.

Note that an alternative but not recommended way of obtaining KPIs is by querying the list of Monitorable services from the service registry and then querying the list of KPI names from the Monitorable services. This way all services which publish KPIs will be returned regardless of whether they do or do not hold the necessary KpiPermission for publishing KPIs. By using the MonitorAdmin to obtain the Monitorables it is guaranteed that only those services will be accessed who are authorized to publish KPIs. It is the responsibility of the MonitorAdmin implementation to check the required permissions and show only those services which pass this check.

12.8.5.1 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.

  The entity which queries a KPI needs to hold KpiPermission for the given target with the read action present.

Returns the KPI object

Throws `IllegalArgumentException` – if the path is invalid or points to a non-existing KPI

`SecurityException` – if the caller does not hold a KpiPermission for the KPI specified by path with the read action present

12.8.5.2 public Monitorable getMonitorable(String id) throws IllegalArgumentException

id the persistent identity of the monitorable service

- Returns the Monitorable service addressed by its PID. For security reasons this method should be used instead of querying the monitorable services from the service registry.

Returns the Monitorable object

Throws `IllegalArgumentException` – if the ID is invalid or points to a non existing monitorable service

12.8.5.3 public Monitorable[] getMonitorables( )

- Returns all the Monitorable services that are currently registered. For security reasons this method should be used instead of querying the monitorable services from the service registry.

Returns the array of Monitorables or null if none are registered

12.8.5.4 public MonitoringJob[] getRunningJobs( )

- Returns the list of currently running Monitoring Jobs.

Returns the list of running jobs

12.8.5.5 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 specifying the number of changes that must happen before a new notification is sent.

- Starts a Monitoring Job with the parameters provided. All specified KPIs must exist when the job is started. 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.

  The entity which initiates a Monitoring Job needs to hold KpiPermission for all the specified target KPIs with the startjob action present. In case of time based jobs, if the permission's action string specifies a minimal sampling interval then the schedule parameter should be at least as great as the value in the action string.

Returns the successfully started job object

Throws `IllegalArgumentException` – if the list of KPI names contains a non-existing KPI or the schedule or count parameters are invalid

`SecurityException` – if the caller does not hold KpiPermission for all the specified KPIs, with the startjob action present, or if the permission does not allow starting the job with the given frequency

12.8.6 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.

12.8.6.1 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

12.8.6.2 public String[] getKpiNames( )

- Returns the list of KPI names that are the targets 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

12.8.6.3 public int getReportCount( )

- Returns the number of 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

12.8.6.4 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

12.8.6.5 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

12.8.6.6 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.

12.8.7 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.

12.8.7.1 public void updated(KPI kpi)

kpi the KPI which has changed

□ Callback for notification of a KPI change.

12.9 References

[25] *SyncML Device Management Tree Description*

[26] *ETSI Performance Management [TS 132 403]*
http://webapp.etsi.org/action/PU/20040113/ts_132403v050500p.pdf

[27] *RFC-2396 Uniform Resource Identifiers (URI): Generic Syntax*
http://www.ietf.org/rfc/rfc2396.txt

13 XML Parser Service Specification

Version 1.0

13.1 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 [31] *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

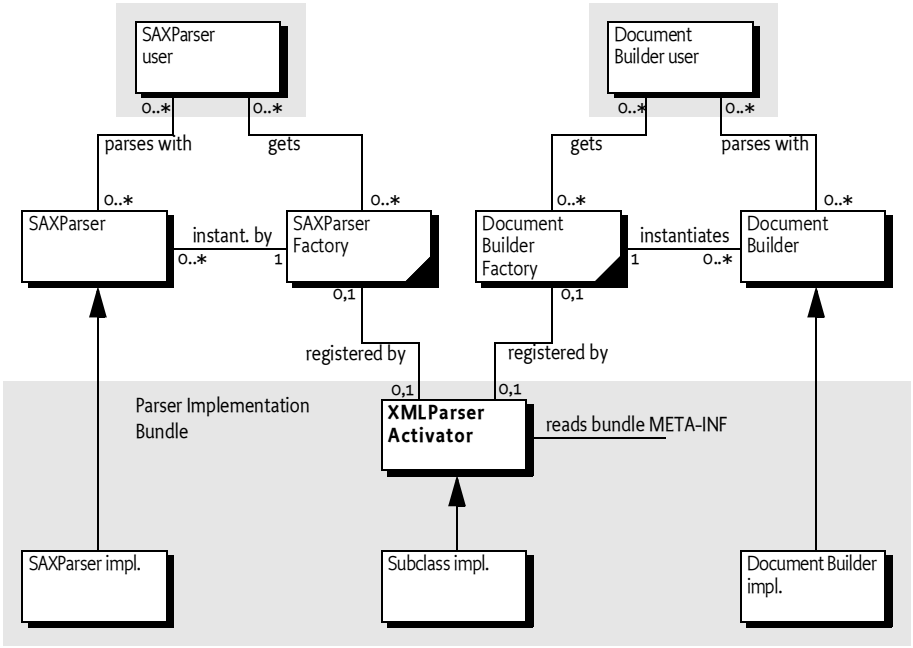
13.1.1 Essentials

- *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

13.1.2 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 57 XML Parsing diagram



13.1.3 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.

13.2 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 [31] *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.

13.3 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.

13.4 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.

13.5 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();
}
```

13.6 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.

13.6.1 JAR Based Services

Its functionality is based on the definition of the [32] *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.

13.6.2 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 285, that describe the instances.

13.6.3 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.

13.7 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 [31] *JAXP* for the exact version dependencies.

This specification is related to related packages as defined in the JAXP 1.1 document. Table 14 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 14 JAXP 1.1 minimum package versions

The Xerces project from the Apache group, [33] *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.

13.8 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.

13.9 org.osgi.util.xml

The OSGi XML Parser service Package. Specification Version 1.0.

13.9.1 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.290])
- javax.xml.parsers.DocumentBuilderFactory(DOMFACTORYNAME[p.290])

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.290] - indicates if this factory supports validating parsers. It's value is a Boolean.
- PARSER_NAMESPACEAWARE[p.290] - 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.

13.9.1.1 public static final String DOMCLASSFILE = "/META-INF/services/javax.xml.parsers.DocumentBuilderFactory"

Fully qualified path name of DOM Parser Factory Class Name file

13.9.1.2 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.

13.9.1.3 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.

13.9.1.4 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.

13.9.1.5 public static final String SAXCLASSFILE = "/META-INF/services/javax.xml.parsers.SAXParserFactory"

Fully qualified path name of SAX Parser Factory Class Name file

13.9.1.6 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.

13.9.1.7 public XMLParserActivator( )

13.9.1.8 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

13.9.1.9 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);

13.9.1.10 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);

13.9.1.11 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`

13.9.1.12 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`

13.9.1.13 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.

13.10 References

[28] *XML*
http://www.w3.org/XML

[29] *SAX*
http://www.saxproject.org/

[30] *DOM Java Language Binding*
http://www.w3.org/TR/REC-DOM-Level-1/java-language-binding.html

[31] *JAXP*
http://java.sun.com/xml/jaxp

[32] *JAR File specification, services directory*
http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html

[33] *Xerces 2 Java Parser*
http://xml.apache.org/xerces2-j

End Of Document