OSGi Service Platform

Release 3
March 2003

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)

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

Digitally signed by OSGi
DN: cn=OSGi, o=Open Service Gateway Initiative, c=US
Date: 2003.03.26 01:55:44 +01'00'

LEGAL TERMS AND CONDITIONS REGARDING SPECIFICATION

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

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

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

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

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

Trademarks

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

Feedback

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

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

OSGi Member Companies

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

OSGi Board and Officers

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

Table Of Contents

Foreword

At the beginning of our fourth year of operation, the Open Services Gateway Initiative is pleased to present our OSGi Service Platform, Release 3 specification. This represents the culmination of another year of cooperative effort by the members of the OSGi alliance that builds on our previous releases and the experience of using those releases to build compelling products and network deployments using the OSGi service delivery model. The army of volunteers who have contributed their time and expertise with the support of the OSGi member companies have made this release and the success of the OSGi alliance possible. We are honored to have such high caliber people involved.

The Open Services Gateway Initiative released the first service platform specification in May, 2000, and the second *OSGi Service Platform specification, Release 2* in October, 2001. Release 2 was published as a book in May, 2002, when we announced the OSGi Compliance Program. With the cooperation of automotive OEMs and the Automotive Multimedia Interface Collaboration (AMI-C), Release 3 will include support for mobile service platforms and applications where data access is handled by a variety of secure interfaces. The Vehicle Expert Group that was formed to work closely with automotive OEMs and service providers has defined these specifications. Release 3 will also have a new reference section including a Reference Architecture and a Remote Management Reference Architecture to help readers understand how the OSGi Service Platform can be used. We have also added a recommended section containing complete specifications of services that are subject to change as more experience is gained with their use. This section includes Jini™ Driver and UPnP™ Service specifications produced by the OSGi Device Expert Group and a Name-space specification. All the other specifications will be maintained with strict backwards compatibility with new OSGi Service Platform releases.

Our original intention was to create a specification to allow services to be remotely deployed onto home network gateways - things like set-top boxes and DSL Modems. The first release of the specification explicitly addressed this market. That specification was extremely successful with many companies creating frameworks compatible with it. Our second release built upon the experiences gained from the first release and introduced methods for improving security, remotely managing service platforms, and making it easier to implement complex applications. OSGi Service Platform, Release 2 became more of a 'horizontal platform', applicable to other markets such as consumer electronics and automotive systems, security products, and mobile phones.

The OSGi alliance exists to create open specifications for the network delivery of managed services to devices in the home, car, and other environments.

The OSGi principles are applicable in any environment where managed lifecycles, long uptimes, and highly resilient, remotely managed platforms are requirements. The automotive industry has added the requirement to keep the applications using that platform current with the latest consumer elec-

tronics systems that may have multiple product life-cycles during the typical ownership period of an automobile.

We've been very careful to not alienate our early adopters, so you'll find very few incompatibilities between this specification and earlier ones. We will continue expanding the specification and developers can rest assured we'll give the same attention to backwards compatibility in the future. This is even more important as other standards organizations are including the OSGi service platform specification as part of their standards.

As said before, we're privileged to have world class people working on this initiative. This document is the product of the OSGi Expert Groups and those people deserve a special pat on the back for their efforts.

Finally, we really do mean the "Open" in our name, so if the OSGi mission is important to you then please come and join us, and together we'll be able to make future releases even better...

So, here it is, the OSGi Service Platform, Release 3. Enjoy...

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

This specification is divided into three sections. The first section contains reference documents. These reference documents provide background information and define the terminology that is used in the remainder of the specifications.

The second section contains the OSGi normative specifications. Future versions of normative specifications will be made fully backward compatible or replaced by a new specification. Normative specifications have a version number starting with 1.

The third and last section contains recommended specifications. The purpose of these specifications is to provide well-defined specifications but allow future specifications to learn from real experiences. Every attempt will be made to keep these specifications backward compatible. However, in certain cases, changes could be made that are not backward compatible.

1.2 What is New In Release 3

The following list details the new specifications that have been added to Release 3.

- *Execution Environment* – In the *Execution Environment Specification* on page 427, the OSGi defines two Execution Environments. One is the minimum requirements defined for test suites and applications that need an absolutely minimal execution environment. The other is adopted from the J2ME Foundation Profile.
- *Reference Architectures* – Two chapters have been included to describe an overall reference architecture and a remote management reference architecture. These architectures define the terminology for the service definitions and place these services and utilities in context. See *Reference Architecture* on page 15 and *Remote Management Reference Architecture* on page 29 for more information.
- *IO Connector Service* – The OSGi specifications have adopted the javax.microedition.io package but added a facility to extend the supported schemes in run-time. See *IO Connector Service Specification* on page 277.
- *Wire Admin Service* – The Wire Admin service provides bundles the opportunity to connect Consumer services to Producer services in run-time, enabling a powerful component model. See *Wire Admin Service Specification* on page 325 for more information.
- *Start Levels* – The Framework API has been extended with a start level service. This service allows the management agent to control the startup and shutdown ordering of bundles. This is defined in the *Start Level Service Specification* on page 137.
- *Measurement* – The Measurement class is a utility for the common problem of handling measurements. It supports a defined unit system and handles measurement errors. It is defined in the *Measurement and State Specification* on page 403.
- *Position* – The Position class is usually used in conjunction with the Wire Admin service. It contains the different aspects of a geographic location. It is detailed in the *Position Specification* on page 421.
- *XML Parser support* – Java 2 specifies a common way of registering XML parsers as extensions. This specification defines a utility to support this mechanism in an OSGi Service Platform. See *XML Parser Service Specification* on page 367 for more information.
- *URL Stream and Content Handlers* – Java allows applications to extend URL Stream and Content Handlers in run-time, but the mechanism used is problematic for an OSGi Service Platform due to the life-cycles that bundles go through. See *URL Handlers Service Specification* on page 155 for more information.
- *Dynamic Import* – Bundles can now indicate that packages should be loaded from other bundles, even though the packages are not explicitly specified in the Import-Package manifest header. This is necessary to support the common Class.forName() idiom. It is specified in *Dynamically Importing Packages* on page 48.
- *Initial Provisioning* – Chapter 26, *Initial Provisioning* on page 529, specifies how the management agent, as defined by the remote management reference architecture, is initially loaded in a service platform.
- *Jini* – The *Jini™ Driver Service Specification* on page 489 details the needed steps and guidelines to use Jini services from an OSGi bundle and how to export OSGi services to Jini communities.
- *UPnP™* – Universal Plug 'n Play supports ad-hoc networking of components from different vendors. The *UPnP™ Device Service Specification* on

page 503 specifies how UPnP devices available in the local network can be used from an OSGi bundle and how a bundle can publish a UPnP device.
- *Indexing* – This document is more extensively indexed than release 2. See the index at the end of this book on page 559.
- An overview of all methods, classes, and packages defined in the OSGi specifications is added in *Method Overview* on page 549.

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 of zero or
              more times, e.g. ( ',' list ) *
?             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 ::= '"' ... '"'
```

Spaces are ignored unless specifically noted.

1.4.2 Object Oriented Terminology

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

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

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

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

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

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

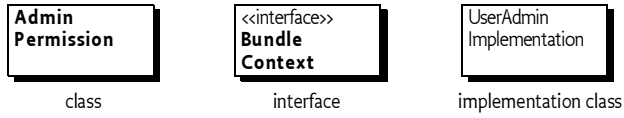
Service objects are registered with the OSGi Framework. Registration consists of the service object, a set of properties, and a list of classes and interfaces implemented by this service object. The classes and interfaces are used for type safety *and* naming. Therefore, it is said that a service object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

1.4.3 Diagrams

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

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

Figure 1 *Class and interface symbol*

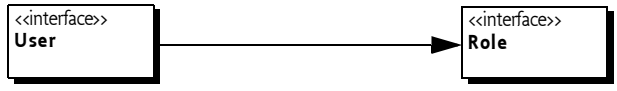


If an interface or class is used as a service object, it will have a black triangle in the bottom right corner.

Figure 2 *Service symbol*

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

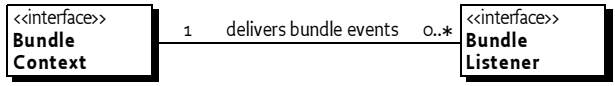
Figure 3 *Inheritance (implements or extends) symbol*



Relations are depicted with a line. The cardinality of the relation is given explicitly when relevant. Figure 4 shows that each (1) BundleContext object is related to 0 or more BundleListener objects, and that each BundleListener object is related to a single BundleContext object. Relations usually have

some description associated with them. This description should be read from left to right and top to bottom, and includes the classes on both sides. For example: "A BundleContext object delivers bundle events to zero or more BundleListener objects."

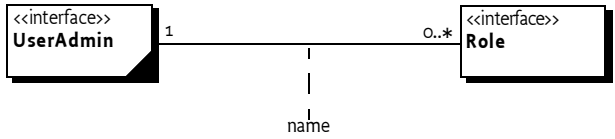
Figure 4 *Relations symbol*



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

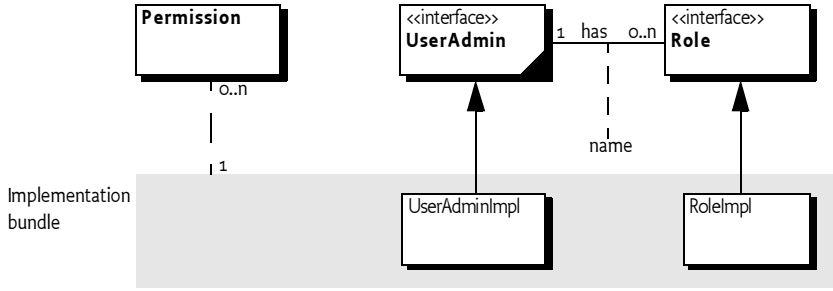
When a relationship is qualified by a name or an object, it is indicated by drawing a dotted line perpendicular to the relation and connecting this line to a class box or a description. Figure 5 shows that the relationship between a UserAdmin class and a Role class is qualified by a name. Such an association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

Figure 5 *Associations symbol*



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

Figure 6 *Bundles*



1.4.4 Key Words

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

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

1.5 The Specification Process

Within the OSGi, specifications are developed by Expert Groups (EG). If a member company wants to participate in an EG, it must sign a Statement Of Work (SOW). The purpose of an SOW is to clarify the legal status of the material discussed in the EG. An EG will discuss material which already has Intellectual Property (IP) rights associated with it, and may also generate new IP rights. The SOW, in conjunction with the member agreement, clearly defines the rights and obligations related to IP rights of the participants and other OSGi members.

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

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

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

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

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

1.6 Version Information

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

New for this specification are:

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

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

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

Table 1 *Packages and versions*

When a component is represented in a bundle, a specification-version is needed in the declaration of the Import-Package or Export-Package manifest headers. Package versioning is described in *Sharing Packages* on page 46.

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

Reference Section

The following section contains documents that are for reference only and are not considered normative.

2 Reference Architecture

Version 1.0

2.1 Introduction

This chapter provides an OSGi reference architecture. This section is neither a specification nor normative. Its purpose is to define a clear and concise terminology as a foundation for the OSGi specifications.

The primary OSGi reference architecture is based on a model where an operator manages a potentially large network of service platforms. It assumes that the service platforms are fully controlled by the operator and are used to run services from many different service providers.

This is, however, only *one* scenario for using the OSGi specifications. Other models might be similar to the deployments of PCs (where the end user has full control over the Service Platform), might be industrial applications (for example, mobile phone base stations where a management center is fully responsible for all aspects), or might be some intermediate model. The wide applicability of the OSGi specifications for networked services makes it impossible to pin-point one model as the final architecture for OSGi. Therefore, some alternate models are discussed in the last sections of this chapter.

The OSGi reference architecture should therefore not be used to restrict possible applications of the OSGi specifications. The OSGi specifications are developed in a cohesive and de-coupled way to allow them to be used in many circumstances, not limited by a single reference architecture.

2.1.1 Essentials

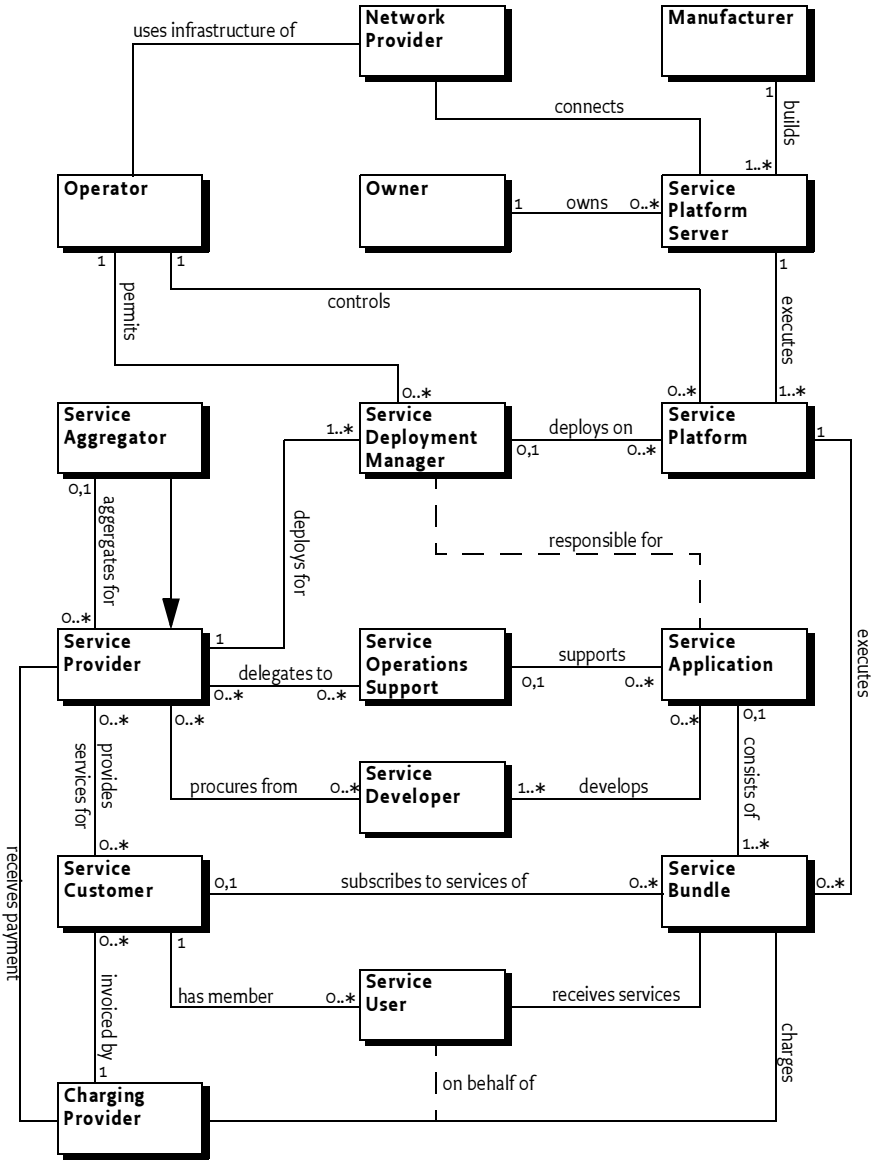
- *Business Driven* – The architecture must be driven from the point of view of the Operator. The Operator needs to make a business case illustrating the management of Service Platforms providing fee based services to end users.
- *Complete* – The architecture must be sufficiently detailed as to allow the vendors to produce robust implementations.
- *Not Constraining* – Service Platforms vary greatly in their capabilities and the network environments in which they operate. The architecture should therefore not overly constrain implementations.
- *Open* – Being a standard and not a design for a specific system, the OSGi reference architecture must consider and support a number of different scenarios.
- *Not Normative* – The reference architecture should not be used as a normative specification.

2.1.2 Entities

- *Service Platform* – A instantiation of a Java VM, an OSGi Framework, and a set of running bundles.

- *Service Platform Server (SPS)* – The hardware that hosts one or more Service Platforms.
- *Operator* – The organization that is in charge of a number of Service Platforms.
- *Service Application* – A suite of bundles, documentation, and support software that together form an application that provides a utility to the Service User.
- *Service User* – The person that receives the benefits of a Service Application.
- *Service Provider* – The organization that procures or develops Service Applications and deploys these applications via a Service Deployment Manager on Service Platforms.
- *Service Deployment Manager (SDM)* – The system that deploys and partially manages the Service Applications of one or more Service Providers.
- *Service Operations Support* – Supporting software and hardware that does not reside on the Service Platform Server but is needed to execute the Service Application.
- *Service Aggregator* – A Service Provider that is responsible for assuring the integrity of service applications from different Service Providers and consolidating them into a single offering.
- *Service Developer* – An organization that develops Service Applications.
- *Manufacturer* – The organization that builds a Service Platform Server.
- *Owner* – The person or organization that has ownership of a Service Platform Server.
- *Charging Provider* – The organization that receives accounting information and that provides a consolidated bill to the Service Customer.
- *Service Platform Identifier* – A unique identity for a Service Platform.
- *Service Customer* – The entity used for billing.
- *Network Provider* – The organization that provides the network connectivity to the Service Platforms.
- *Certification Authority* – An organization that can manage certificates used to authenticate systems, individuals, and organizations.

Figure 7 Architecture Diagram



2.2 Entity Descriptions

It should be noted that most entities are roles. One particular person or organization can have several roles simultaneously. For example, the same organization might act as a *Service Provider*, *Service Platform Operator* and *Service Deployment Manager*.

2.2.1 Service Platform

The primary function of the *Service Platform* is to manage the execution life-cycle of *Service Applications*. Service Applications consist of bundles that the Service Platform is capable of dynamically loading, activating, deactivating, updating, and unloading.

2.2.2 Service Platform Server

The Service Platform Server (SPS) hosts one or more Service Platforms. It can be equipped with very specialized hardware suited for a specific application domain. For example, it can contain an MPEG 4 decoder chip or it can house any number of processors executing any number of Service Platforms.

2.2.3 Operator

The primary responsibility of the *Operator* is to control who is allowed to deploy services to the Service Platform in question i.e. control which *Service Deployment Managers* are allowed to manage the particular Service Platform. In addition to this, the Operator can also manage other functions related to a specific Service Platform instance.

2.2.4 Service Application

A *Service Application* is a set of bundles that collectively implement a specific function used within a Service Platform.

2.2.5 Service User

The *Service User* is defined to be an entity, possibly human, that does one or more of the following:

- Initiates the necessary business events required to ensure that a specific Service Application is deployed on a specific Service Platform.
- Interacts with a specific Service Application during its execution.

Service Users are end users. They might or might not interact directly with the Service Platform Server as they use it. When incurring charges for using a service, a Service User must be associated with a *Service Customer*. For example, when a Service User watches a Pay-TV movie, the associated Service Customer incurs the charges.

2.2.6 Service Provider

The *Service Provider* represents a business related entity. The Service Provider supplies the necessary means to provide the business related support of a specific Service Application. The Service Provider is also responsible for delegating the tasks of service deployment and service operation management to the Service Deployment Manager and Service Operations Support.

2.2.7 Service Deployment Manager

The *Service Deployment Manager (SDM)* acts on behalf of the Service Provider. The SDM manages all issues related to the life-cycle of Service Applications that are external to the Service Platform.

2.2.8 Service Operations Support

The *Service Operations Support* (SOS) acts on behalf of the Service Provider. The SOS is responsible for the systems that operate in conjunction with the Service Applications. For example, if a Service Application is acting as a Web client accessing a Web server that provides support to the application, the Web server is part of the responsibilities of the SOS.

2.2.9 Service Developer

A *Service Developer* writes Service Applications. In general, they are considered to be the same as the Service Provider. However, there are cases where the distinction is important. For example, a software company could specialize in generic service application development and sell software suites to Service Providers.

2.2.10 Service Aggregator

A *Service Aggregator* is a Service Provider that consolidates Service Applications provided by other Service Providers into one distinct offering and resolves dependencies and conflicts between different Service Applications.

A Service Aggregator is similar to a general contractor who consolidates the Service Applications of multiple subcontractors.

There is little in this architecture that is specific to a Service Aggregator. Except as noted, information about a Service Provider also applies to a Service Aggregator.

2.2.11 Manufacturer

The Manufacturer is responsible for integration of the SPS hardware, operating system, Java VM, and OSGi Framework. There are two general cases for the Service Platform Server:

- The generic SPS
- The special-purpose SPS

The generic SPS is manufactured to be managed by any Operator. The special-purpose SPS is manufactured for a specific Operator. The latter case is simpler because keys, certificates, addresses, etc. can be installed at the factory, eliminating the need for a secure protocol that allows these items to be assigned at install time.

2.2.12 Owner

The *Owner* owns the Service Platform Server. Owner is used loosely here so it applies to a lessor as well.

In most cases, the Owner is assumed to be a Service User. However, there are some cases where this might not be true. A landlord might buy and install a single SPS to be shared by the tenants in an apartment building. Each tenant has the role of Owner, but the multiple tenants might not trust each other.

In many cases, the entity performing the Owner role can also be the Operator.

2.2.13 Service Platform Identifier

The *Service Platform Identifier* (SPI) is a character string that is assigned at some point in time before the Service Platform is running, also known as *staging*. This identifier must be stored in the Service Platform Server so that it is available to the Service Platform software when it starts. The identifier must also be made available to the Owner. The identifier can be printed on the exterior of the Service Platform Server either on its box or the accompanying literature. If the manufacturer uses a smart card to hold the Service Platform Identifier, it could, for example, be printed on the smart card.

The identifier must be unique across all service platforms, and it should be formatted according to the following scheme:

```
SPI     := domain ':' tail
domain  := alpha ( alpha | digit | '_' ) *
tail    := alpha ( alpha | digit | [-+/._] ) *
```

The first part of the string is a domain identifier, which defines the structure and format of the rest of the identifier. The domain identifier and the tail are delimited by a colon (":"). The domain identifier can indicate anything, such as manufacturer, phone number, or some other existing name-space. For example:

```
MSISDN:+46706066934
VIN:123456789-345678
ACME:SerNo987654321-0
```

The Service Platform Identifier can be permanent or changeable. An example of a permanent identifier is a serial number built into the Service Platform Server and used under the assumption that only one Service Platform executes on that server.

A changeable identifier might be one that can be created during the initial provisioning of the SPS, either by the Service Platform itself or by an external entity. A changeable identifier might be based on a permanent one, such as a serial number built into the Service Platform Server. For example, a server serial number appended with the ordering number of the service platform can be a suitable identifier.

It is important for the Service Platform Identifier to be unique and for the Operator to understand the nature (permanent or otherwise) of the Service Platform Identifier.

2.2.14 Service Customer

The *Service Customer* is the entity that is responsible for paying charges that are incurred using services. A Service Customer is associated with a Charging Provider.

A Service Customer can have one or more Service Users associated with it. For example, in a single household, the telephone subscription is usually associated with a single person in that household, but all family members can use the telephone. In this case, the other family members would be Service Users.

2.2.15 Network Provider

The Network Provider provides and manages wide-area network connectivity between the Service Platform and outside parties. Outside parties include the Operator and other Service Providers. In the case where the Service Platform is connected via the Internet, the Network Provider is assumed to include the Internet Service Provider (ISP) functionality.

A Service Platform that connects to the Internet using DSL might have (in the aftermath of deregulation) five companies responsible for the various layers: the phone company responsible for the copper wires to the home, the DSL service provider responsible for the ATM connectivity, the ISP responsible for Internet connectivity, the Operator responsible for operation of the Service Platform, and finally the other Service Provider(s) responsible for the value added service(s) running.

This model applies to both IP and non-IP network layers and to both continuous and intermittent availability. Every effort is made to avoid making assumptions that the network service is Internet (IP) or that it is continuously available.

2.2.16 Charging Provider

The *Charging Provider* performs the following roles in the OSGi reference architecture:

- Grants (or denies), the Service Platform the ability to perform a specific service for a specific Service User,
- Stores and/or forwards charging events coming from the Service Platform,
- Invoices the Service Customer,
- Settles bills with the Service Provider.

The Charging Provider is a recognized entity in the OSGi architecture but does not play a role in any of the specifications in this document.

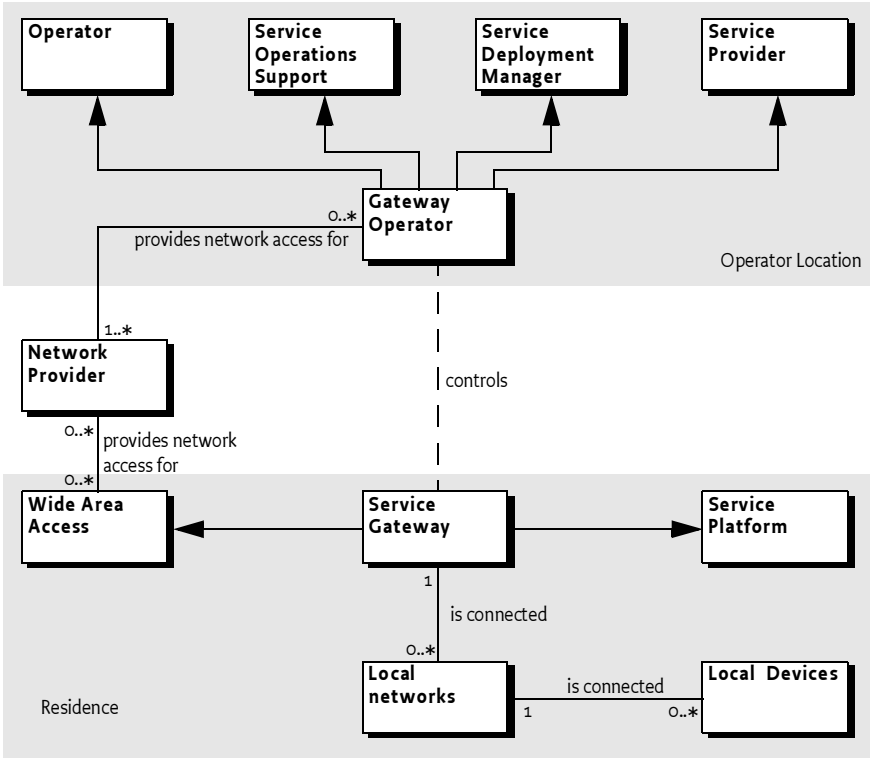
2.2.17 Certification Authority

A *Certification Authority* is a trusted third-party organization or company that issues digital certificates used to create digital signatures and public-private key pairs. The Certification Authority guarantees that the individual granted the unique certificate is, in fact, who he or she claims to be. Typically, a Certification Authority makes these certificates available in some common database (usually a directory.) Certificate Authorities must be trusted in order for their certificates to be meaningful. The Certificate Authority is part of a Public Key Infrastructure (PKI). A very large PKI can also include a registration authority or even a local registration authority that does actual physical verification of the credentials.

2.3 The Service Gateway Model

Great care has been taken to make the OSGi specifications as general as possible. However, an important application area of the specifications is large scale systems with residential gateways. A typical description of such an architecture is described in [5] *An Electronic Service's enabler*. The basic model is illustrated in Figure 8 *Possible Gateway model*.

Figure 8 *Possible Gateway model*



Where the Service Platform Server (SPS) represents a communication gateway, the gateway Operator is a party that normally assumes the following roles:

- *Service Provider* – for managing the gateway WAN connection and managing the local networks. The latter might span from only ensuring that the local devices have proper connectivity to controlling all operations related to these devices.
- *Operator* – for the Service Platforms in the SPS.
- *Service Deployment Manager* – for the services running on the Service Platforms in the SPS.
- *Service Operations Support* – for the services running on the Service Platforms in the SPS.

2.3.1 System environment for a service gateway

A Service Platform Server is often an IP gateway between a WAN and one or more local networks. In addition, it is also able to execute arbitrary services and is therefore called a service gateway. The separate tasks of a service gateway are a Service Platform and an Internet Protocol gateway. This model also supports service gateways with non-IP based WAN connections through a Non-IP WAN Connection.

By separating the IP gateway from the Service Platform it is *simpler* to apply existing solutions for network and security management of IP gateways to service gateways.

It should be noted that the division of the service gateway is logical and not physical. A typical service gateway probably uses a single processor that executes both the Service Platform and the IP gateway functions (and/or Non-IP WAN connection functions).

2.3.1.1 IP Gateway

The IP gateway deals with forwarding incoming IP traffic from WAN, LAN, and Service Platforms to the appropriate next hop, WAN, LAN or Service Platform. Firewall functions are typically located in the IP gateway. The IP gateway can, from a functional perspective, be viewed as a *standard* access router.

2.3.1.2 Non-IP Local Network Adaptor

Communication with non-IP based local networks is performed through an adaptor for the specific type of network. This adaptor can include functions such as firewall, address translation (NAT), and more.

2.3.1.3 Local Devices Network

A local bus is used for communication with local devices. The local bus technologies differ depending on various factors such as the following:

- Type of service gateway (e.g. residential, vehicle)
- Market (e.g. US, Europe, or Asia)

Typical local bus technologies are as follows:

- European Installation Bus (EIB)
- Ethernet
- Wireless Ethernet, WiFi
- Firewire, IEEE 1394B
- Media Oriented Systems Transport (MOST)

A local device is something that is connected to the local buses, sends and/or receives information from the Service Platform, and/or sends and/or receives information through the IP gateway.

Devices that are able to communicate through the IP gateway are separated from devices that must communicate through the Service Platform. Security solutions for these two cases are somewhat different and, therefore, the distinction is important.

2.3.1.4 Wide Area Network

A service gateway is typically connected to a Wide Area Network (WAN). Typically, a service gateway is connected to just one WAN, but multiple WANs connected to a single service gateway are not excluded. Typical WAN technologies are as follows:

- DSL
- Ethernet
- Cellular phone networks (e.g. GSM, AMPS)

The WAN is most commonly used to carry Internet Protocol (IP) traffic. A WAN with the capability of carrying IP traffic is called an IP WAN.

Some service gateways do not have IP connectivity or only have IP connectivity under special circumstances. For example, a Service Gateway in a car might communicate with its Gateway Operator using wireless protocols (e.g. GSM SMS).

2.3.1.5 Mediation of IP with non-IP

Operators, Service Providers, and Service Aggregators are typically connected to the Service Platform using the Internet or a private IP network. For the Operator to communicate with Service Platforms, which are not capable of communicating using IP, some kind of mediation of the communication is required. A WAP (Wireless Access Protocol used by telephones) gateway is one example.

2.4 Other Models

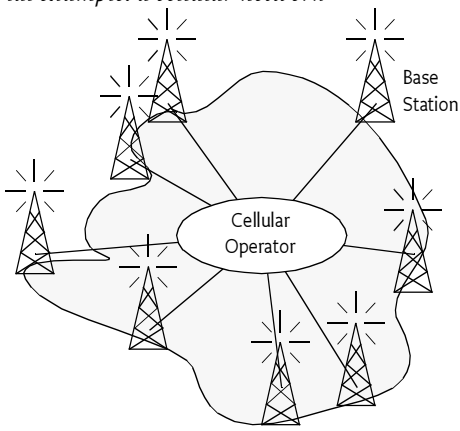
The OSGi reference architecture is designed to handle the complexity of a Operator based network with external service providers. This does not mean that the OSGi reference architecture is limited to this model. Care has been taken to ensure that the architecture, and thus the services that are developed for this architecture, can be used in many different situations. The following sections discuss possible applications of the architecture that differ from the typical service gateway model.

2.4.1 Industrial Model

The OSGi networked services model is applicable when computers need to be remotely managed but there is no need for an open Service Provider model.

A typical example is a cellular network. In cellular networks, base stations provide coverage to subscribers throughout the country. Base stations are complex computers that run large and complex software programs. Today, most of these systems are proprietary and use proprietary or standardized management protocols for remote management. The size of the networks usually requires procurement of services from multiple vendors, creating unavoidable inter operability problems.

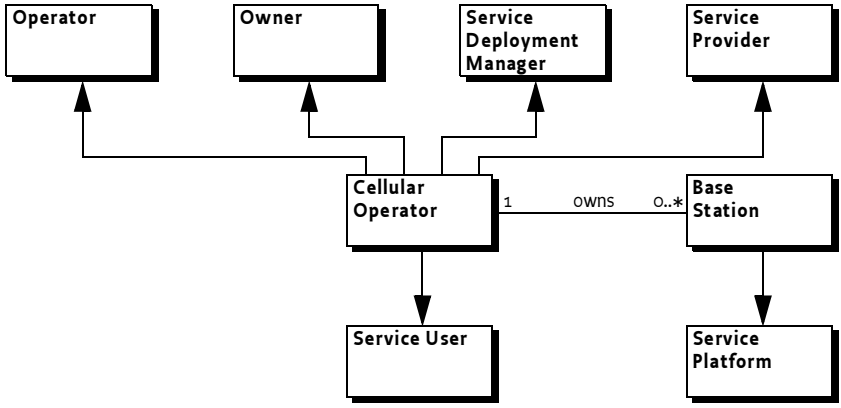
Figure 9 *Industrial example: a cellular network*



Implementing the OSGi specifications in this way enables the use of standardized software (from different vendors) for many functions that are written today with proprietary software. The OSGi architecture of managed services enables the cellular management system to manage the network much more coherently than if such a architecture is not used. It would, for example, be possible to define bundles that implement operator optimized management policies local in the base station, something that is hard to do with today's systems.

The mapping of an industrial cellular network model to the OSGi reference architecture is depicted in Figure 10.

Figure 10 *Industrial Model Mapping Example*



The industrial model does *not* require all software to be written in Java, something that is often not feasible for economic and technical reasons. Java can be restricted to the control layers where performance is less of an issue. The core of the applications can be based on native code. For this reason, the OSGi bundle specifically allows native code to be embedded in the JAR file (even for multiple operating systems).

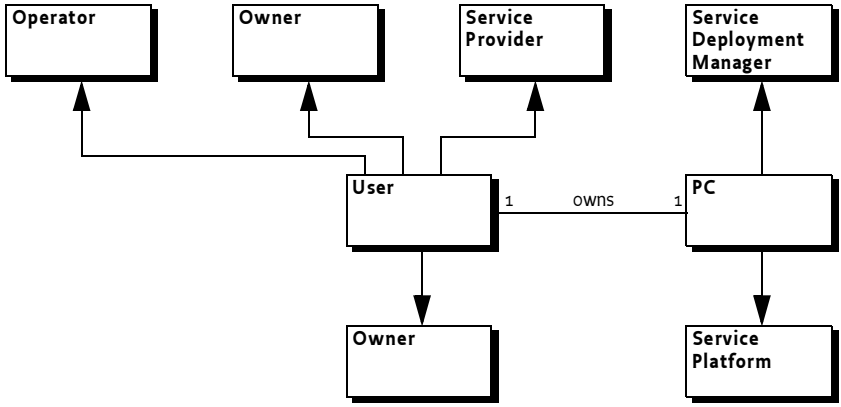
The industrial model is applicable whenever many computers are networked and need to be managed from one or more centralized management centers, as in the following examples:

- Assembly lines
- Firewalls in remote offices
- Point of sales terminals
- Energy Management systems for large buildings
- Alarm Systems
- Fleet management

2.4.2 Self-Managed Model

Another feasible implementation of the OSGi reference architecture is the self-managed model. This model is similar to the model that is used with computers (PCs) in the home; the Service User assumes the role of the Operator and manages the systems. In this model, the Service Platform contains the Service Deployment Manager (probably as a bundle) that allows self management, for example, via a Web based interface.

Figure 11 Self Managed Model Mapping Example



This model is applicable for applications written in Java running on Service Platform Servers within the same local network as the Operator. The number of applications is small enough to not require a management server, as shown in the following examples:

- Router/Firewall box
- WiFi base station
- Printer
- Private Branch Exchange (PBX)

2.4.3 Virtual Gateway Model

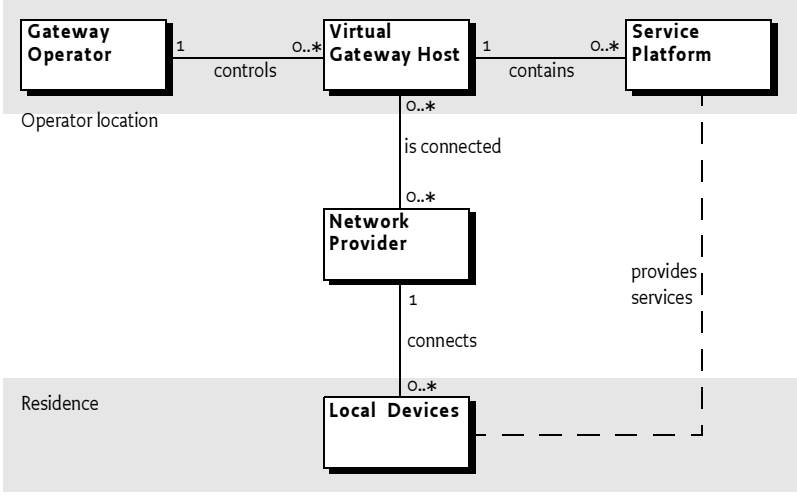
Many models have the implicit assumption that the Server Platform Server (SPS) is placed in a home or on the customer premises. This is, however, not necessary. It is possible to place the SPS in the network in an Operator controlled environment and provide the services via the network. This architecture has the following advantages:

- More Operator control over the hardware.

- Less truck rolls to homes where residents are not always available.
- Resources are more efficiently used when they can be shared.
- Easier to secure in all aspects.

The disadvantage of this model is that it makes it harder to connect local devices because all services must be delivered over the network and cannot have a local component. Such a model is described in [6] *Telia's Service Delivery Solution for the Home.*

Figure 12 *Virtual Gateway Mapping Example*



2.5 Security

Overall security is only as strong as the weakest link. Therefore, it must be possible to implement each applicable OSGi specification in a secure way. All specifications must assume that their implementations will be used in a hostile environment, such as being connected to the Internet, where security is paramount.

This implies that all specifications must contain a security analysis and guidelines section that discusses security implications. In cases where security is not applicable, it should be argued why it is not applicable in that case.

2.6 References

[4] *X.200 OSI Reference Model*
http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.200-199407-I

[5] *An Electronic Service's enabler*
http://www.ericsson.com/about/publications/review/1999_01/files/1999015.pdf

[6] *Telia's Service Delivery Solution for the Home*
http://www.osgi.org/news/osgi_news/research_ieee_artikel_020415.pdf

3 Remote Management Reference Architecture

Version 1.0

3.1 Introduction

The remote management reference architecture defines a comprehensive management model that leverages the possibilities of a Service Platform to allow management systems and Server Platforms to inter-operate in a non-proprietary way. The remote management reference architecture is derived from the OSGi *Reference Architecture* on page 15.

This chapter details a remote management reference architecture to be used by other OSGi specifications. This management reference architecture is *not* normative, nor is it possible to claim compliance to it. The purpose of this architecture is to define a consistent terminology for use throughout the specifications.

This remote management reference architecture should not be used to restrict possible applications of the OSGi specifications. The OSGi specifications are developed in a cohesive and de-coupled way to allow them to be used in many different circumstances, not limited by this reference architecture.

This reference architecture is not complete and might be extended in future releases of the OSGi specifications.

3.1.1 Essentials

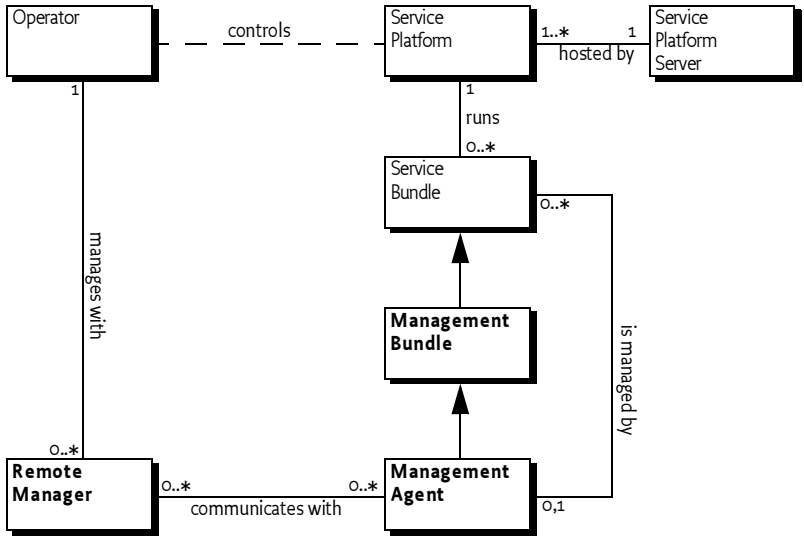
- *Business Driven* – While remote management is intended to relieve the end user from any administrative duties, the customer for a remote management solution is an Operator. A remote management architecture must address the business needs of the Operator.
- *Complete* – The remote management architecture must be sufficiently specified to allow managementsystem vendors to produce robust and interoperable implementations.
- *Not Constraining* – Because Service Platforms vary greatly in their capabilities and in the network environments in which they operate, the remote management architecture should not overly constrain the implementation of a remote management solution.
- *Owned* – It must be possible for an Operator to establish and maintain control over the management of a Service Platform. In other words, a competing Operator cannot take control of a Service Platform without consent of the owner.
- *Upgradeable* – It must be possible for the Operator to upgrade or replace the Management Agent.

- *Provisioned* – There must be a set of reasonable alternatives for provisioning remote management for a Service Platform. While it may be acceptable to provision a Service Platform during manufacturing, it must be possible to complete provisioning after manufacturing. A very complete set of provisioning alternatives includes a means of completing the provisioning after the end user has taken possession of the Service Platform Server with a minimum of user interaction.
- *Compatible* – In whatever language the services and OSGi framework are written, the primary life-cycle management characteristics must be similar to the current Java based Framework.
- *Managed* – Existing and future *local* life-cycle management and configuration services should be leveraged for remote management.
- *Extendable* – It must be possible to support different management protocols that may be added incrementally, even on already deployed platforms.
- *Granular* – Remote management may be implemented as a collection of services, each useful in itself.
- *Open* – Because it is a standard and not a design for a specific system, the remote management architecture of OSGi must consider and support a number of different scenarios.
- *Initial Provisioning* – The remote management reference architecture must provide a standard for staging a platform, binding to an operator, and downloading an initial management agent.

3.1.2 Entities

- *Remote Manager* – The Operator's system(s) that are communicating with the Service Platforms to provide remote management.
- *Management Agent* – A set of one or more bundles that run on the Service Platform and communicate with the Remote Manager to provide management of the Service Platform. In certain cases, a Management Agent can also be provided by the System Bundle.
- *Management Bundle* – An OSGi Bundle that has AdminPermission and can control the life-cycle and configuration of other bundles.
- *Initial Provisioning* – The process in which a Service Platform is provisioned with a Management Agent.

Figure 13 *Remote Management Entities*



3.2 Scope

The concept of a Service Platform includes zero administration usage by the Service Users, allowing them to make decisions on a service subscription level rather than worry about details like software versions, needed revisions of drivers, conflicting dependencies between different software packages, etc.

The Service Platform should be designed to have a high availability and deliver services with a specified level of determinism. For Service Platforms operated by an Operator, these requirements necessitate *remote management* and monitoring of the Service Platform services and devices.

Remote Management, in this context, may include (As adapted from [7] *X.700 Management framework for OSI/CCITT applications*):

- Bundle life-cycle management
- Configuration management
- Performance management
- Fault management
- Accounting management
- Security management

Remote management is fundamentally a process involving two distinct roles, with one party acting as agent (recipient) and the other acting as manager (sender) of management commands. There are two distinct roles: one that is *managed* and one that *manages*.

The entity on the Service Platform that receives the management instructions is called the *Management Agent*. The entity responsible for managing the Service Platform is the *Operator*. The component of the Operator's infrastructure that provides this management will be called the *Remote Manager*. The reference architecture combines the Management Agent and Remote Manager in the Service Deployment Manager.

3.2.1 Remote Manager

The *Remote Manager* is the system that exposes the remote management interface of the Service Platform to the Operator. Because different management protocols present different views of remote management of a platform, only high-level requirements for the capabilities of the Remote Manager are specified.

Primarily, the Remote Manager is expected to perform a certain set of operations on the Service Platform. Those operations may be classified as:

- *Bundle life-cycle management* – Installing, starting, updating, stopping and uninstalling bundles.
- *Security management* – Setting the Permissions for bundles and handling User data on the Service Platform.
- *Configuration management* – Setting configuration data.
- *Fault management.* – Running diagnostics and correcting problems.
- *Accounting management* – Collecting accounting information and sending them to the Charging Provider (The OSGi currently has no specifications related to accounting).
- *Performance management* – Optimizing resource usage on the Service Platform (The OSGi currently has no specifications related to performance management).

No restrictions are made here as to what kinds of systems may behave as a Remote Manager or Management Agent. Thus, peer-to-peer management is theoretically possible, with either peer playing the Remote Manager role at times and the Management Agent role at other times.

3.2.2 Management Agent

A *Management Agent* is implemented with a *Management Bundle* (which may be the System Bundle). The Management Agent should accept communications from the Remote Manager or may initiate them.

The Management Agent is a Management Bundle and must, therefore, have `AdminPermission`. There may be more than one Management Agent. Each Management Agent may be configured to communicate with different Remote Managers and may use different protocols. Assuming each Management Agent has administrator permissions, Management Agents have equal power to affect the operations of the Service Platform.

In order to establish and maintain the security of the Service Platform, the Management Agent must be involved in any action that results in bundle download or installation, configuration, permissions management, or other important changes to the Service Platform. Ideally, the designated Manage-

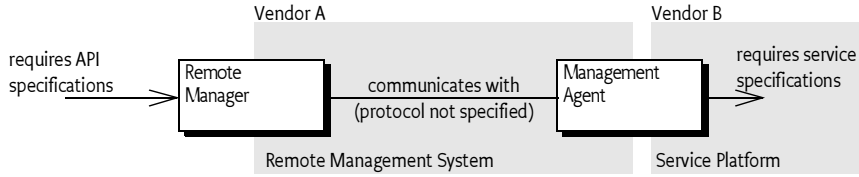
ment Agent is the only means by which management actions can be affected from the outside. In that way, the Management Agent can be an effective gatekeeper, allowing only Remote Managers with proper credentials to change the operation of the Service Platform.

3.3 Communications

A key aspect of the Remote Management reference architecture is that it does *not* define a protocol between the Remote Manager and the Management Agent, nor does it make any statements about what occurs between these entities. The reference architecture is only concerned with the external interfaces of the Management Agent and the Remote Manager.

The purpose of the reference architecture is to allow interoperability between any proprietary management system and any Service Platform. This is achieved by installing a management vendor-specific Management Agent in the Service Platform that can communicate with a proprietary management system. Requirements on the communications between the Remote Manager and Management Agent are therefore not necessary. They have become implementation details that are left up to the implementers of the Management Agent and the Remote Manager.

Figure 14 *Abstracting the Communications*



The following sections discuss some of the aspects of implementing a Remote Manager and a Management Agent.

3.3.1 Connectivity

Service Platforms are deployed in a wide variety of situations. Though generic IP connectivity will be available in many of those situations, it should not be assumed to be always present. A significant number of installations will have low-bandwidth, intermittent connectivity. OSGi Management Systems that need to be widely applicable should take this into account.

3.3.2 Protocols

There are already both standard and proprietary management protocols in the industry. Some are suited to managing certain kinds of Service Platforms over certain types of networks. New standards may be needed in order to have an open solution that is suitable for wireless wide area networks (GSM, GPRS, CDMA, etc.).

Management System vendors, or independent software developers, can develop Management Agents that adopt applicable standards. These Management Agents can then be used by existing management systems that support these protocols.

3.3.3 Secure Connections

Communications may be secure or not, depending upon the operating environment of the Service Platform and Operator. It is strongly recommended that OSGi solutions should have at least the following:

- Mutual authentication
- Message integrity checking
- Confidentiality

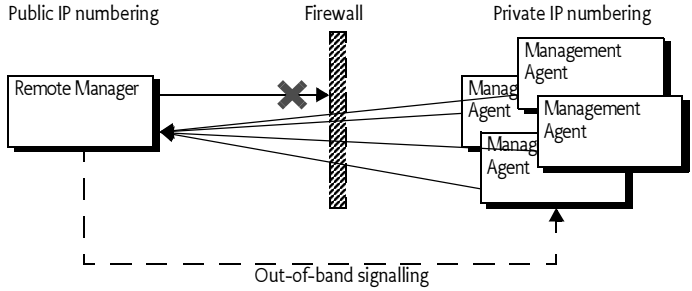
3.3.4 Network Restrictions

There will be many networks that do not allow direct access from the Remote Manager to a Service Platform. Network firewalls, network address translation (NAT), non-IP networks, and other constructions often make it impossible to directly contact a Service Platform from the Operator premises. In contrast, the Service Platform can usually contact the Remote Manager when needed. This means that there is often a need for an out-of-band signalling mechanism to ask the Service Platform to initiate contact with the Remote Manager. This is depicted in Figure 15.

For example, in a vehicle, a message, like the GSM Short Message Service (SMS), could be used to request the vehicle to contact the Remote Manager. However, many cases exist where such an out-of-band signalling mechanism is not available or is prohibitively expensive for normal day-to-day operations.

Remote management implementations must address this issue with care.

Figure 15 *Network Restrictions*



3.4 Initial Provisioning

The term *Initial Provisioning* refers to all the steps required to enable remote management. This may include downloading and starting a Management Agent, configuring this bundle with information about the Remote Manager, establishing sufficient permissions for the Management Bundle, and establishing a security association between the Management Agent and the Remote Manager.

Starting at production, the Service Platform may be provisioned for a certain Operator who uses a certain kind of Remote Manager.

A Service Platform without provisioning may be provisioned on its way to the operational site where it is installed. Other Service Platforms must be adapted to a specific Operator and Remote Manager when they are activated at their operational site.

Part of provisioning a Remote Manager and Management Agent for management is to establish the security association between them. If public key cryptography is used, it implies that each has either the other's certificates or is provisioned with a suitable Certification Authority. For other communication schemes, other information is provisioned. For example, if a symmetric key authentication and digest algorithm (such as MD5) is used, a shared secret is provisioned before management is established.

3.5 Security

Consider all the communications between a Service Platform and other devices and systems. The vast majority of these communications may be classified as *normal operation*, meaning that services on the platform are communicating with devices on local networks and remote systems on wide area networks in order to accomplish their goals. For example, a media service running on a platform may be interacting with a remote media server from which it receives content and a local device on which the content is rendered. The media service may also communicate with a *Charging Provider*. The Charging Provider may even reconfigure the media service on the platform if the user has signed up for a different level of service.

Some of these communications may be secure (confidential, with great attention paid to mutual authentication), and some may be clear, unencoded transmissions. Even broadcasting of these communications is considered part of normal application functions.

That these communications cannot disrupt normal operation of the service platform is taken as an article of faith in the OSGi core platform design. However, misuse of the core platform features can easily compromise the security of the Service Platform.

For example, bundles with AdminPermission must be few. Errant assignment of AdminPermission severely compromises the platform.

By contrast, a Management Agent must have AdminPermission in order to manage. The communications between a Management Agent and any remote system must be carefully examined. Mutual authentication, confidentiality, and message integrity checks should be used.

The remote management architecture specifies neither an abstract management protocol nor any transport protocols that may be used to carry a management protocol. Consequently, most of the security considerations for remote management can only be addressed through recommendations such as those mentioned here in the previous paragraphs.

In addition to the authentication of peers, it is also desirable to encrypt the response data so that it may contain privacy related data.

3.6 References

[7] *X.700 Management framework for OSI/CCITT applications*
http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.700

[8] *The Java Security Architecture for JDK 1.4*
http://java.sun.com/j2se/1.4/docs/guide/security

Normative Section

The following section contains OSGi normative specifications. Every attempt will be made to make future versions of these specifications to be backward compatible with these specifications.

4 Framework Specification

Version 1.2

4.1 Introduction

The Framework forms the core of the OSGi Service Platform specifications. It provides a general-purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable service applications known as *bundles*.

OSGi-compliant devices can download and install OSGi bundles, and remove them when they are no longer required. Installed bundles can register a number of services that can be shared with other bundles under strict control of the Framework.

The Framework manages the installation and update of bundles in an OSGi environment in a dynamic and scalable fashion, and manages the dependencies between bundles and services.

It provides the bundle developer with the resources necessary to take advantage of Java's platform independence and dynamic code-loading capability in order to easily develop, and deploy on a large scale, services for small-memory devices.

Equally important, the Framework provides a concise and consistent programming model for Java bundle developers, simplifying the development and deployment of services by de-coupling the service's specification (Java interface) from its implementations. This model allows bundle developers to bind to services solely from their interface specification. The selection of a specific implementation, optimized for a specific need or from a specific vendor, can thus be deferred to run-time.

A consistent programming model helps bundle developers cope with scalability issues – critical because the Framework is intended to run on a variety of devices whose differing hardware characteristics may affect many aspects of a service implementation. Consistent interfaces insure that the software components can be mixed and matched and still result in stable systems.

As an example, a service developed to run on a high-end device could store data on a local hard drive. Conversely, on a diskless device, data would have to be stored non-locally. Application developers that use this service can develop their bundles using the defined service interface without regard to which service implementation will be used when the bundle is deployed.

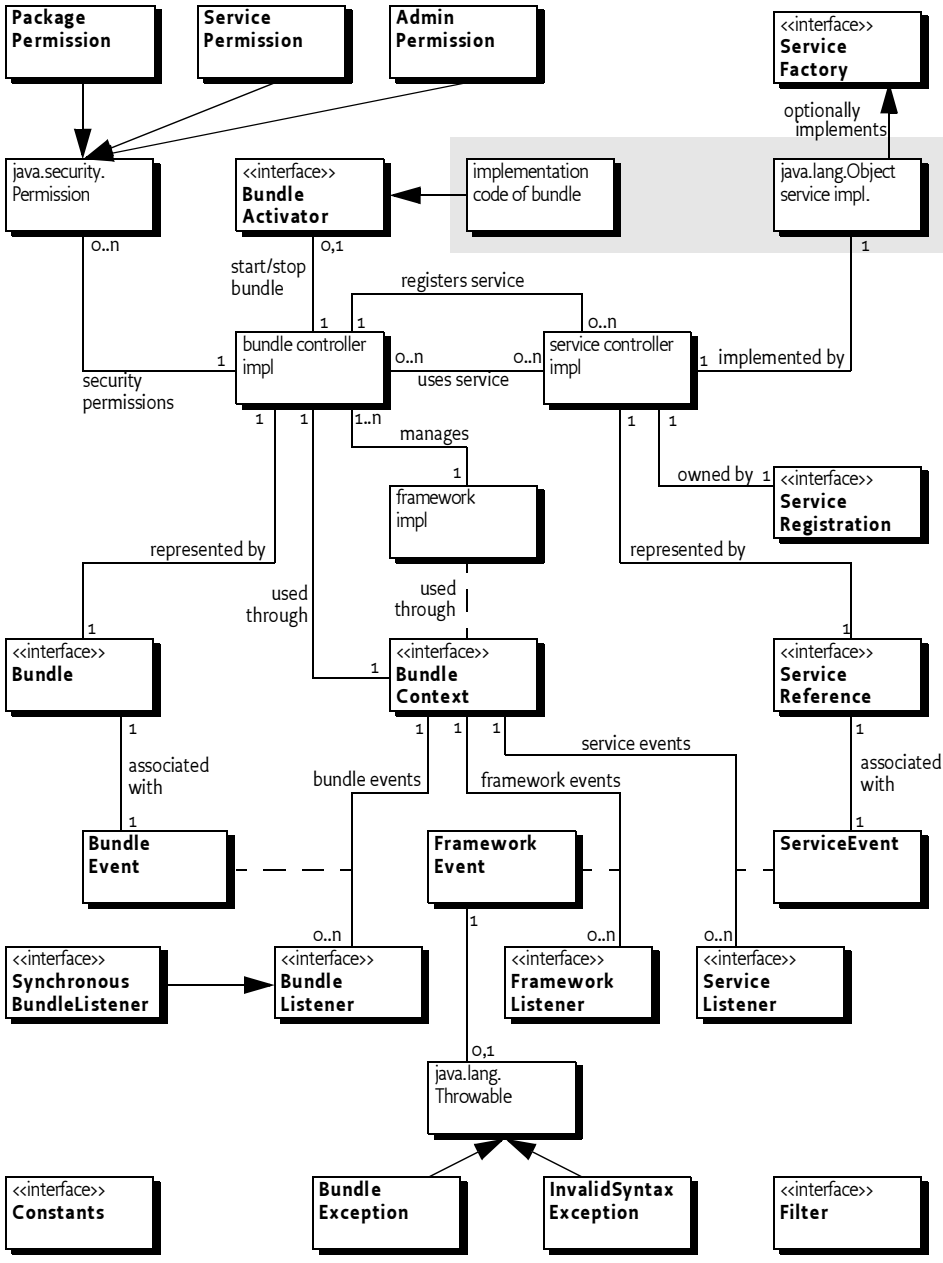
The Framework allows bundles to select an available implementation at run-time through the Framework service registry. Bundles register new services, receive notifications about the state of services, or look up existing services to adapt to the current capabilities of the device. This aspect of the Framework makes an installed bundle extensible after deployment: new bundles can be installed for added features or existing bundles can be modified and updated without requiring the system to be restarted.

The Framework provides mechanisms to support this paradigm which aid the bundle developer with the practical aspects of writing extensible bundles. These mechanisms are designed to be simple so that developers can quickly achieve fluency with the programming model.

4.1.1 Entities

Figure 16 on page 41 provides an overview of the classes and interfaces used in the org.osgi.framework package. It shows the relationships between the different Framework entities. This diagram is for illustrative purposes only. It can show details that may be implemented in different ways.

Figure 16 *Class Diagram* org.osgi.framework



4.2 Bundles

In the OSGi Service Platform, bundles are the only entities for deploying Java-based applications. A bundle is comprised of Java classes and other resources which together can provide functions to end users and provide components called services to other bundles, called *services*. A bundle is deployed as a Java ARchive (JAR) file. JAR files are used to store applications and their resources in a standard ZIP-based file format.

A bundle is a JAR file that:

- Contains the resources to implement zero or more services. These resources may be class files for the Java programming language, as well as other data such as HTML files, help files, icons, and so on.
- Contains a manifest file describing the contents of the JAR file and providing information about the bundle. This file uses headers to specify parameters that the Framework needs in order to correctly install and activate a bundle.
- States dependencies on other resources, such as Java packages, that must be available to the bundle before it can run. The Framework must resolve these packages prior to starting a bundle. See *Sharing Packages* on page 46.
- Designates a special class in the bundle to act as Bundle Activator. The Framework must instantiate this class and invoke the start and stop methods to start or stop the bundle respectively. The bundle's implementation of the BundleActivator interface allows the bundle to initialize (for example, registering services) when started, and to perform cleanup operations when stopped.
- Can contain optional documentation in the OSGI-OPT directory of the JAR file or one of its sub-directories. Any information in this directory must not be required to run the bundle. It can, for example, be used to store the source code of a bundle. Management systems may remove this information to save storage space in the OSGi Service Platform.

Once a bundle is started, its functionality is provided and services are exposed to other bundles installed in the OSGi Service Platform.

4.2.1 The System Bundle

In addition to normal bundles, the Framework itself is represented as a bundle. The bundle representing the Framework is referred to as the *system bundle*.

Through the system bundle, the Framework may register services that may be used by other bundles. Examples of such services are the Package Admin and Permission Admin services.

The system bundle is listed in the set of installed bundles returned by BundleContext.getBundles(), although it differs from other bundles in the following ways:

- The system bundle is always assigned a bundle identifier of zero (0).
- The system bundle getLocation method returns the string: "System Bundle", as defined in the Constants interface.

- The system bundle cannot be life-cycle-managed like normal bundles. Its life-cycle methods must behave as follows:
- *start* – Does nothing because the system bundle is already started.
  - *stop* – Returns immediately and shuts down the Framework on another thread.
  - *update* – Returns immediately, then stops and restarts the Framework on another thread.
  - *uninstall* – The Framework must throw a BundleException indicating that the system bundle cannot be uninstalled.

  See *Framework Startup and Shutdown* on page 79 for more information about the starting and stopping of the Framework.
- The system bundle's Bundle.getHeaders method returns a Dictionary object with implementation-specific manifest headers. For example, the system bundle's manifest file should contain an Export-Package header declaring which packages are to be exported by the Framework (for example, org.osgi.framework).

4.3 Manifest Headers

A bundle can carry descriptive information about itself in the manifest file that is contained in its JAR file under the name of META-INF/MANIFEST.MF.

The Framework defines OSGi manifest headers such as Export-Package and Bundle-Activator, which bundle developers can use to supply descriptive information about a bundle. Manifest headers must strictly follow the rules for manifest headers as defined in [17] *Manifest Format*.

All manifest headers are optional and any standard manifest headers not specified have no value by default (except for Bundle-Classpath that has dot ('.', \u002E) as default when no value is specified).

All manifest headers that may be declared in a bundle's manifest file are listed in Table 2, "Manifest Headers," on page 44.

A Framework implementation must:

- Process the main section of the manifest. Individual sections of the manifest may be ignored.
- Ignore unrecognized manifest headers. Additional manifest headers may be defined by the bundle developer as needed.
- Ignore unknown attributes on OSGi-defined manifest headers.

4.3.1 Retrieving Manifest Headers

The Bundle interface defines a method to return manifest header information: getHeaders(). This method returns a Dictionary object that contains the bundle's manifest headers and values as key/value pairs.

This method requires AdminPermission because some of the manifest header information may be sensitive, such as the packages listed in the Export-Package header.

The getHeaders method must continue to provide the manifest header information after the bundle enters the UNINSTALLED state.

4.3.2 Manifest Headers

All specified manifest headers are listed in Table 2.

Header	Sample	Description
Bundle-Activator	com.acme.fw.Activator	The name of the class that is used to start and stop the bundle. See *Starting Bundles* on page 59.
Bundle-Category	osgi, test, nursery	A comma separated list of category names.
Bundle-ClassPath	/jar/http.jar,.	A comma separated list of JAR file path names (inside the bundle) that should be searched for classes and resources. The period ('.') specifies the bundle itself. See *Bundle Classpath* on page 51.
Bundle-ContactAddress	2400 Oswego Road Austin, 74563 TX	Contact address if it is necessary to contact the vendor.
Bundle-Copyright	OSGi (c) 2002	Copyright specification for this bundle.
Bundle-Description	Network Firewall	A short description of this bundle.
Bundle-DocURL	http:/www.acme.com/Firewall/doc	A URL to documentation about this bundle.
Bundle-Name	Firewall	Name for this bundle. This should be a short name and should contain no spaces.
Bundle-NativeCode	/lib/http.DLL ; osname = QNX ; osversion = 3.1	A specification of native code contained in this bundle's JAR file. See *Loading Native Code Libraries* on page 53.
Bundle-Required ExecutionEnvironment	CDC-1.0/Foundation-1.0	Comma separated list of execution environments that must be present on the Service Platform. See *Execution Environment* on page 52.
Bundle-UpdateLocation	http://www.acme.com/Firewall/bundle.jar	If the bundle is updated, this location should be used (if present) to retrieve the updated JAR file.
Bundle-Vendor	OSGi	A text description of the vendor.
Bundle-Version	1.1	The version of this bundle.
DynamicImport-Package	com.acme.plugin.*	Comma separated list of package names that should be dynamically imported when needed. See *Dynamically Importing Packages* on page 48.

Table 2 *Manifest Headers*

Header	Sample	Description
Export-Package	org.osgi.util.tracker	Comma separated list of package names (with optional version specification) that can be exported. See *Exporting Packages* on page 47.
Import-Package	org.osgi.util.tracker, org.osgi.service.io; specification-version=1.4	Comma separated list of package names (with optional version specification) that must be imported. See *Importing Packages* on page 48

Table 2 *Manifest Headers*

4.4 The Bundle Name-space

This section addresses the issues related to class loading in the Framework and the details necessary to implement a Framework.

A *classloader* (ClassLoader object) loads classes into the Java Virtual Machine. When such classes refer to other classes or resources, they are found through the *same* classloader. This classloader may load the class itself or delegate the loading to another classloader. This approach effectively creates a name-space for classes. A class is uniquely identified by its fully qualified name and the classloader that created it. This implies that a class can be loaded multiple times from different classloaders. These classes, though they have the same name, are not compatible.

4.4.1 Bundles and Classloaders

Each bundle installed in the Framework and resolved must have a classloader associated with it (Frameworks may have multiple classloaders per bundle). This classloader provides each bundle with its own name-space, to avoid name conflicts, and allows package sharing with other bundles. The bundle's classloader must find classes and resources in the bundle by searching the bundle's classpath as specified by the Bundle-Classpath header in the bundle's manifest. See *Bundle Classpath* on page 51 for more information on this header.

Bundles collaborate by sharing objects that are an instance of a mutually agreed class (or interface). This class must be loaded from the *same* classloader for both bundles. Otherwise, using the shared object will result in a ClassCastException. Therefore, the Framework must ensure that all importers of a class in an exported package use the same classloader to load that class or interface.

For example, a bundle may register a service object under a class com.acme.C with the Framework service registry. It is crucial that the bundle that created the service object ("Bundle A") and the one retrieving it from the service registry ("Bundle B") share the same class com.acme.C of which

the service object must be an instance. If Bundle A and Bundle B use different classloaders to load class com.acme.C, Bundle B's attempt to cast the service object to its version of class com.acme.C would result in a ClassCastException.

A bundle's classloader must also set the ProtectionDomain object for classes loaded from the bundle, as well as participate in requests to load native libraries selected by the Bundle-NativeCode manifest header.

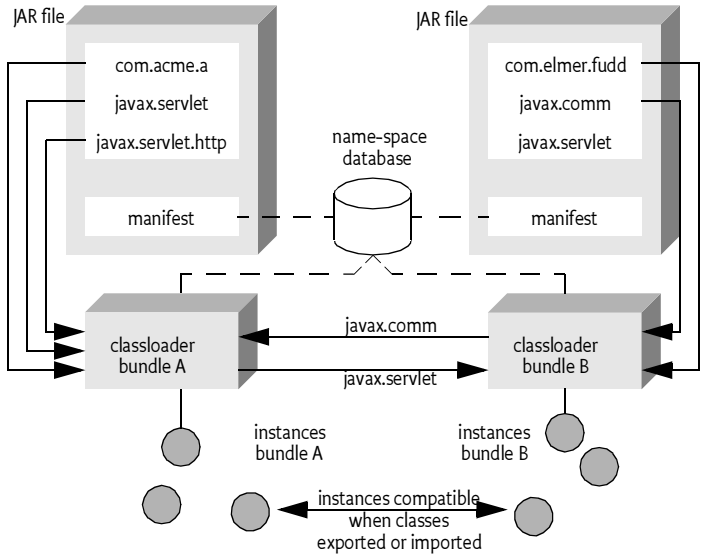
The classloader for a bundle is created between installing and resolving the bundle.

4.4.2 Sharing Packages

A bundle may offer to export all the classes and resources in a package by specifying the package names in the Export-Package header in its manifest. For each package offered for export, the Framework must choose one bundle that will be the provider of the classes and resources in that package to all bundles which import that package, or other bundles which offer to export the same package.

Selecting a single package among all the exporters ensures that all bundles share the same class and resource definitions. If a bundle does not participate in the sharing of a package – in other words, the bundle does not have an Export-Package, Import-Package, or DynamicImport-Package manifest header referencing the package – then attempts by the bundle to load a class or resource from the package must not search the shared package space. Only the system classpath and the bundle's JAR file are searched for such a package.

Figure 17 *Package and class sharing*



Package sharing for a bundle is established during the process of *resolving* the bundle. A bundle must only participate in sharing packages if the bundle can be successfully resolved. A bundle that is not resolved must neither export nor import packages. A bundle must have the necessary PackagePermission to participate in the sharing of a package.

A bundle declares the resources it offers to provide to other bundles using Export-Package manifest headers, and declares the resources it needs using Import-Package manifest headers. The DynamicImport-Package header allows a bundle to specify packages that are imported when a package is first needed.

4.4.3 Exporting Packages

The Export-Package manifest header allows a bundle to export Java packages to other bundles, exposing the packages to other bundles.

The Framework must guarantee that classes and resources in the exported package's name-space are loaded from the exporting bundle. Additionally, the package's classes and resources must be shared among bundles that import the package. See *Importing Packages* on page 48.

If more than one bundle declares the same package in its Export-Package manifest header, the Framework controls the selection of the exporting bundle. The Framework must select for export the bundle offering the highest version of the declared package.

In order to export a package, a bundle must have PackagePermission[EXPORT,<package>].

The Export-Package manifest header must conform to the following syntax:

```
Export-Package ::=
   package-description
   ( ',' package-description )*

package-description ::=
   package-name ( ';' parameter )*

package-name ::=
   <fully qualified package name>

parameter  ::=
   attribute '=' value

attribute  ::= token
value      ::= token | quoted-string
```

The only package-description parameter recognized by the Framework is the attribute specification-version. Its string value must conform to the semantics described in the [15] *The Java 2 Package Versioning Specification.*

As an example, the following Export-Package manifest header declares that the bundle provides all classes defined by version 2.1 of the javax.servlet and javax.servlet.http packages.

```
Export-Package: javax.servlet;
      specification-version="2.1",
   javax.servlet.http;
      specification-version="2.1"
```

4.4.4 Importing Packages

The Import-Package manifest header allows a bundle to request access to packages that have been exported by other bundles.

The Framework must guarantee that while a bundle is resolved, the bundle is only exposed to one version of a package it has imported.

The fully qualified package name must be declared in the bundle's Import-Package manifest header for all packages a bundle needs, except for package names beginning with:

java.

In order to be allowed to import a package (except for packages starting with java.), a bundle must have PackagePermission[EXPORT|IMPORT, <package>]. See PackagePermission for more information.

The Import-Package manifest header must conform to the following syntax:

```
Import-Package ::=
   package-description
   (  ','  package-description )*

package-description =
   package-name ( ';' parameter )*

package-name  =
   <fully qualified package name>

parameter     = attribute '=' value
attribute     = token
value         = token | quoted-string
```

The only package-description parameter recognized by the Framework is the attribute specification-version. Its string value must conform to the semantics described in the [15] *The Java 2 Package Versioning Specification*.

As an example, the following Import-Package manifest header requires that the bundle be resolved against the javax.servlet package version 2.1 or above:

```
Import-Package: javax.servlet;
   specification-version="2.1"
```

4.4.5 Dynamically Importing Packages

The DynamicImport-Package manifest header should be used when the bundle cannot beforehand define a fixed set of packages to import because this information is only known at run-time.

For example, the Class.forName idiom (where a class is loaded by giving a String object with the name of the class) is used heavily in legacy and non-OSGi applications to connect to classes that are not linked in but designated by configuration information. One case is the Java Media Framework (JMF). JMF uses system properties to specify class names for applicable coders/decoders (codec). JMF uses Class.forName to instantiate user defined codecs.

In the default case, the Class.forName method must only look in the calling bundle or in any of the imported packages from other bundles. The nature of Class.forName is that the package is usually not known when the bundle is created; it comes from run-time configuration information or from information otherwise dynamically obtained. When the bundle needs to import a package that it could not foresee when it was created, it should specify the DynamicImport-Package manifest header.

The syntax of DynamicImport-Package manifest header is as follows:

```
DynamicImport-Package ::==
   package-name ( ',' package-name ) * | '*'

package-name ::= <fully qualified package name>
           | <partial package name> '.*'
```

Package names may end in a wildcard ('.*'), meaning all packages that start with this name. For example. if the DynamicImport-Package header contains org.osgi.*, packages like org.osgi.service.io and org.osgi.impl.service.wireadmin must match the header. A single '*' matches all packages and allows a bundle to import any exported package.

This manifest header must be consulted by the bundle's classloader when:

- A class or resource needs to be loaded
- This resource is not on the classpath, and
- It is not already imported

If this header contains a package name that matches (including wildcard matching) the class' (or resource's) package name, then the classloader must try to import this package as if it was imported during bundle resolving. This includes the necessary PackagePermission[IMPORT|EXPORT, <package>] checks as well as establishing the package import dependencies for the Package Admin service.

There must be no noticeable difference between a bundle that statically imported a package (via Import-Package or Export-Package) and a bundle that has dynamically imported a package.

The DynamicImport-Package header must not be consulted when the bundle is installed and then resolved. Packages that are indicated in this header must not be required to be exported during resolving.

Caution is advised when the dynamic import specification matches packages contained in the bundle's JAR file. Dynamic Import must take precedence over classes and resources provided by the bundle's JAR file.

This implies that when a package is contained in the bundle's JAR file but might also be loaded dynamically, a resource or class might be loaded from the bundle's JAR file *before* another bundle had the opportunity to export the package (this can actually result in split packages). It is therefore recommended to avoid dynamically importing packages that are also available from the bundle's JAR file.

4.4.5.1 Dynamic Import Example

The ACME company has wrapped JMF in a bundle. Plug-ins for JMF are sold separately. However, they do not want other companies to provide plug-ins for their JMF bundle. They therefore require plug-ins to come from their own package name-space (com.acme.*).

The bundle containing JMF requires the following manifest header:

```
DynamicImport-Package: com.acme.*
```

A bundle containing codecs should export the packages where the codec appears in:

```
Export-Package: com.acme.mp3, com.acme.wave
```

When the JMF now tries to load the codec class com.acme.mp3.MP3Codec, the JMF bundle's classloader(s) must import the com.acme.mp3 package dynamically.

4.4.5.2 Dynamic Import and versioning

It is impossible to specify a version with a dynamic import header because it is the purpose of this header to allow the import of unknown packages. It is therefore important that this header is used only for scenarios where the packages are not sensitive to versions. If the packages are known, the Import-Package header should be used.

4.4.6 Importing a Lower Version Than Exporting

Exporting a package does not imply that the exporting bundle will actually use the classes it offers for export. Multiple bundles can offer to export the same package; the Framework must select only one of those bundles as the exporter.

A bundle will implicitly import the same package name and version level as it exports, and therefore a separate Import-Package manifest header for this package is unnecessary. If the bundle can function using a lower specification version of the package than it exports, then the lower version can be specified in an Import-Package manifest header.

4.4.7 Code Executed Before Started

Packages exported from a bundle are exposed to other bundles as soon as the bundle has been resolved. This condition could mean that another bundle could call methods in an exported package *before* the bundle exporting the package is started.

4.4.8 Recommended Export Strategy

Although a bundle can export all its classes to other bundles, this practice is discouraged except in the case of particularly stable library packages that will need updating only infrequently. The reason for this caution is that the Framework may not be able to promptly reclaim the space occupied by the exported classes if the bundle is updated or uninstalled.

Bundle designs that separate interfaces from their implementations are strongly preferred. The bundle developer should put the interfaces into a separate Java package to be exported, while keeping the implementation classes in different packages that are not exported.

If the same interface has multiple implementations in multiple bundles, the bundle developer can package the interface package into all of these bundles; the Framework must select one, and only one, of the bundles to export the package, and the interface classes must be loaded from that bundle. Interfaces with the same package and class name should have exactly the same signature. Because a modification to an interface affects all of its callers, interfaces should be carefully designed and remain backward compatible once deployed.

4.4.9 Bundle Classpath

Intrabundle classpath dependencies are declared in the Bundle-Classpath manifest header. This declaration allows a bundle to declare its internal classpath using one or more JAR files that are contained in the bundle's JAR file. For example, a bundle's JAR file could contain servlet.jar and cocoon.jar as entries. Both entries need to be part of the bundle's classpath. The Bundle-Classpath manifest header specifies these embedded JAR files.

The Bundle-Classpath manifest header is a list of comma-separated file names. A file name can be either dot ('.') or the path of a JAR file contained in the bundle's JAR file. The dot represents the bundle's JAR file itself.

Classpath dependencies must be resolved as follows:

- If a Bundle-Classpath header is not declared, the default value of dot ('.') is used, which specifies the bundle's JAR file.
- If a Bundle-Classpath manifest header is declared and dot ('.') is not an element of the classpath, the bundle's JAR file must not be searched. In this case, only the JAR files specified within the bundle's JAR file must be searched.

The Bundle-Classpath manifest header must conform to the following syntax:

```
Bundle-Classpath ::=  path ( ',' path )*

path ::= <path name of nested JAR file with
     "/"-separated components>  |  '.'
```

For example, the following declaration in a bundle's manifest file would expose all classes and resources stored in the JAR file, but also all classes and resources defined in servlet.jar, to the bundle:

```
Bundle-Classpath: .,
   lib/servlet.jar
```

The Framework must ignore missing files in the Bundle-Classpath headers. However, a Framework should publish a Framework Event of type ERROR for each file that is not found in the bundle's JAR with an appropriate message.

4.5 Execution Environment

A bundle that is restricted to one or more Execution Environments must carry a header in its manifest file to indicate this dependency. This header is Bundle-RequiredExecutionEnvironment. The syntax of this header is a list of comma separated names of Execution Environments.

```
Bundle-RequiredExecutionEnvironment ::=
   ee-name ( ',' ee-name )*

ee-name ::= <defined execution environment name>
```

For example:

```
Bundle-RequiredExecutionEnvironment: CDC-1.0/Foundation-1.0,
  OSGi/Minimum-1.0
```

If a bundle includes this header in the manifest then the bundle must only use methods with signatures that are a proper subset of all mentioned Execution Environments.

The Bundle-RequireExecutionEnvironment header indicates a pre-requisite to the Framework. If a bundle with this header is installed or updated, the Framework must verify that the listed execution environments match the available execution environments during the installation of the bundle. When the pre-requisite cannot be fulfilled, the Framework must throw a BundleException during installation with an appropriate message.

The OSGi Execution Environments are defined in *Execution Environment Specification* on page 427.

4.5.1 Naming of Execution Environments

Execution Environments require a proper name so that they can be identified from a Bundle's manifest as well as provide an identification from a Framework to the bundle of what Execution Environments are implemented. Names consist of any set of characters except whitespace characters and the comma character (' ,', or \u002C). The OSGi has defined a number of Execution Environments. See *Execution Environment Specification* on page 427.

The naming scheme is further based on J2ME configuration and profile names. There is no clear definition for this naming scheme but the same type of names are used in different specifications.

The J2ME scheme uses a configuration and a profile name to designate an execution environment. The OSGi naming combines those two names into a single Execution Environment name.

There already exist a number of Execution Environments from J2ME that are likely to be supported in Service Platform Servers. The value for the Execution Environment header must be compatible with these specifications.

A J2ME Execution Environment name is defined as a combination of a configuration and a profile name. In J2ME, these are 2 different System properties. These properties are:

```
microedition.configuration
microedition.profile
```

For example, Foundation Profile has an Execution Environment name of CDC-1.0/Foundation-1.0. Table 3 on page 53, contains a number of examples.

Name	Description
CDC-1.0/Foundation-1.0	Equal to J2ME Foundation Profile
OSGi/Minimum-1.0	OSGi EE that is a minimal set that allows the implementation of an OSGi Framework.
JavaEmbedded-1.2	Java Embedded
JavaCard	Java Card
CLDC-1.0/MIDP-1.0	MIDP
PersonalJava-1.2	Personal Java
J2EE-1.2	Java 2 EE
J2SE-1.3	Java 2 SE

Table 3 *Sample EE names*

The org.osgi.framework.executionenvironment property from BundleContext.getProperty(String) must contain a comma separated list of Execution Environment names implemented on the Service Platform.

4.6 Loading Native Code Libraries

If a bundle has a Bundle-NativeCode manifest header, the bundle should contain native code libraries that must be available for the bundle to execute. When a bundle makes a request to load a native code library, the findLibrary method of the caller's classloader must be called to return the file path name in which the Framework has made the requested native library available.

The bundle must have the required RuntimePermission[loadLibrary.<library name>] in order to load native code in the OSGi Service Platform.

The Bundle-NativeCode manifest header must conform to the following syntax:

```
Bundle-NativeCode ::=
   nativecode-clause ( ',' nativecode-clause ) *

nativecode-clause ::=  nativepaths ( ';' env-parameter )*
```

```
nativepaths  ::=  nativepath ( ';' nativepath )*

nativepath   ::= </ separated path>

env-parameter ::= (  processordef |  osnamedef  |
                     osversiondef |  languagedef )

processordef      ::= 'processor' '=' value
osnamedef         ::= 'osname'    '=' value
osversiondef      ::= 'osversion' '=' value
languagedef       ::= 'language'  '=' value

value     ::= token  | quoted-string
```

The following is a typical example of a native code declaration in a bundle's manifest:

```
Bundle-NativeCode: /lib/http.DLL ;
   /lib/zlib.dll ;
      osname       = Windows95 ;
      osname       = Windows98 ;
      osname       = WindowsNT ;
      processor    = x86 ;
      language     = en ;
      language     = se ,
   /lib/solaris/libhttp.so ;
      osname       = Solaris ;
      osname       = SunOS ;
      processor    = sparc,
   /lib/linux/libhttp.so ;
      osname       = Linux ;
      processor    =  mips
```

If a Bundle-NativeCode clause contains duplicate env-parameter entries, the corresponding values must be OR'ed together. This feature must be carefully used because the result is not always obvious. This is highlighted by the following example:

```
// The effect of this header
// is probably not the intended effect!
Bundle-NativeCode: /lib/http.DLL ;
     osname       = Windows95 ;
     osversion    = 3.1 ;
     osname       = WindowsXP ;
     osversion    = 5.1
```

The previous example implies that the native library will load on Windows XP 3.1 and later, which was probably not intended. The single clause should be split up when the expected effect is desired:

```
Bundle-NativeCode: /lib/http.DLL ;
     osname       = Windows95 ;
     osversion    = 3.1,
   /lib/http.DLL ;
```

```
osname       = WindowsXP ;
osversion    = 5.1
```

If multiple native code libraries need to be installed on one platform, they must be specified in the same clause for that platform.

4.6.1 Native Code Algorithm

In the description of this algorithm, [*x*] represents the value of the Framework property *x* and ~= represents the match operation. The match operation is a case insensitive comparison. The manifest header should contain the generic name for that property but the Framework should attempt to include aliases when it matches. (See *Environment Properties* on page 63). If a property is not an alias, or has the wrong value, the Operator should set the appropriate system property to the generic name or to a valid value (System properties with this name override the Framework construction of these properties). For example, if the operating system returns version 2.4.2-kwt, the Operator should set the system property org.osgi.framework.os.version to 2.4.2.

The Framework must select the native code clause selected by the following algorithm:

1. Select only the native code clauses for which the following expressions all evaluate to true.
   - osname ~= [org.osgi.framework.os.name]
   - processor ~= [org.osgi.framework.processor]
   - osversion <= [org.osgi.framework.os.version] or osversion is not specified
   - language ~= [org.osgi.framework.language] or language is not specified
2. If no native clauses were selected in step 1, a BundleException is thrown, terminating this algorithm.
3. The selected clauses are now sorted in the following priority order:
   - osversion: osversion in descending order, osversion not specified
   - language: language specified, language not specified
   - Position in the Bundle-NativeCode manifest header: lexical left to right.
4. The first clause of the sorted clauses from step 3 must be used as the selected native code clause.

If a selected native code library cannot be found in the bundle's JAR file, then the bundle installation must fail.

4.7 Finding Classes and Resources

Framework implementations must follow the rules defined in this section regarding class and resource loading to create a predictable environment for bundle developers.

A bundle's classloader responds to requests by the bundle to load a resource or class. The bundle's classloader must use a delegation model. Upon a request to load a resource or class, the following classloaders must be searched for the first occurrence of the class or resource, in the following order:

1. The system classloader.
2. The classloader of the bundle that exports the shared package to which the resource belongs and that package is imported.
3. The bundle's own classloader. The bundle is searched in the order specified in the Bundle-Classpath manifest header. See *Bundle Classpath* on page 51.

A class loaded from a bundle must always belong to that bundle's ProtectionDomain object.

4.7.1 Resources

In order to have access to a resource in a bundle, appropriate permissions are required. A bundle must always be given the necessary permissions by the Framework to access the resources contained in its JAR file (these permissions are Framework implementation dependent), as well as permission to access any resources in imported packages.

When findResource is called on a bundle's classloader, the caller is checked for the appropriate permission to access the resource. If the caller does not have the necessary permission, the resource is not accessible and null must be returned. If the caller has the necessary permission, then a URL object to the resource must be returned. Once the URL object is returned, no further permission checks are performed when the contents of the resource are accessed. The URL object must use a scheme defined by the Framework implementation, and only the Framework implementation must be able to construct such URL objects of this scheme. The external form of this URL object must be defined by the implementation.

A resource in a bundle may also be accessed by using the Bundle.getResource method. This method calls getResource on the bundle's classloader to perform the search. The caller of Bundle.getResource must have AdminPermission.

4.7.2 Automatically Importing java.*

All bundles must dynamically import packages which wildcard match java.* (this must be automatic and should not be specified with a DynamicImport-Package manifest header).

Since bundles are not required to specify packages beginning with java. in the Import-Package manifest header, this allows java.* packages to be provided by either the normal system classpath or via another bundle with no knowledge beforehand of from where the package originates.That is, it allows the classpath to be extended with java packages by bundles.

In support of the above, the Framework must give all bundles the implied permission: PackagePermission[IMPORT,"java.*"] to allow them to successfully import packages starting with java..

4.8 The Bundle Object

For each bundle installed in the OSGi Service Platform, there is an associated Bundle object. The Bundle object for a bundle can be used to manage the bundle's life-cycle. This is usually done with a Management Agent.

4.8.1 Bundle Identifier

The bundle identifier is unique and persistent. It has the following properties:

- The identifier is of type long.
- Once its value is assigned to a bundle, that value must not be reused for another bundle, even if the original bundle is reinstalled.
- Its value must not change as long as the bundle remains installed.
- Its value must not change when the bundle is updated.

The Bundle interface defines a getBundleId() method for returning a bundle's identifier.

4.8.2 Bundle Location

The bundle location is the location string that was specified when the bundle was installed. The Bundle interface defines a getLocation() method for returning a bundle's location attribute.

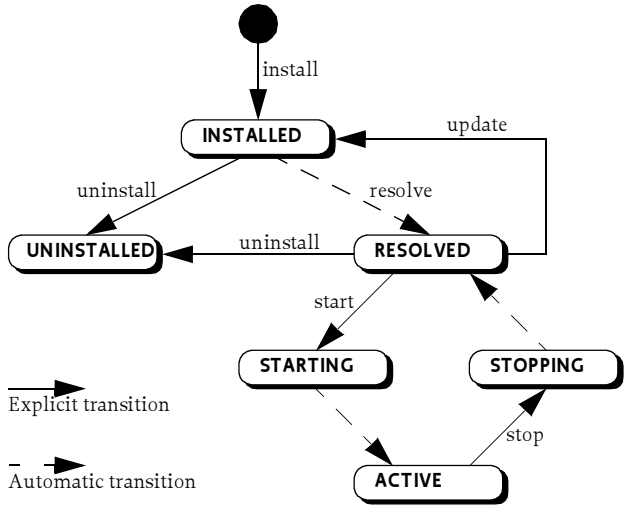
A location string uniquely identifies a bundle and must not change when a bundle is updated.

4.8.3 Bundle State

A bundle may be in one of the following states:

- INSTALLED – The bundle has been successfully installed. Native code clauses must have been validated.
- RESOLVED – All Java classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped.
- STARTING – The bundle is being started, and the BundleActivator.start method has been called and has not yet returned.
- STOPPING – The bundle is being stopped, and the BundleActivator.stop method has been called and has not yet returned.
- ACTIVE – The bundle has successfully started and is running.
- UNINSTALLED – The bundle has been uninstalled. It cannot move into another state.

Figure 18 *State diagram Bundle*



When a bundle is installed, it is stored in the persistent storage of the Framework and remains there until it is explicitly uninstalled. Whether a bundle has been started or stopped must be recorded in the persistent storage of the Framework. A bundle that has been persistently recorded as started must be started whenever the Framework starts until the bundle is explicitly stopped. The Start Level service influences the actual starting and stopping of bundles. See *Start Level Service Specification* on page 137.

The Bundle interface defines a getState() method for returning a bundle's state.

Bundle states are expressed as a bit-mask to conveniently determine the state of a bundle. A bundle can only be in one state at any time. The following code sample can be used to determine if a bundle is in the STARTING, ACTIVE, or STOPPING state:

```
if ((b.getState() & (STARTING | ACTIVE | STOPPING) != 0)
   ...
```

4.8.4 Installing Bundles

The BundleContext interface, which is given to the Bundle Activator of a bundle, defines the following methods for installing a bundle:

- installBundle(String) – Installs a bundle from the specified location string (which should be a URL).
- installBundle(String,InputStream) – Installs a bundle from the specified InputStream object.

Every bundle is uniquely identified by its location string. If an installed bundle is using the specified location, the installBundle methods must return the Bundle object for that installed bundle and not install a new bundle.

The installation of a bundle in the Framework must be:

- *Persistent* – The bundle must remain installed across Framework and Java VM invocations until it is explicitly uninstalled.
- *Atomic* – The install method must completely install the bundle or, if the installation fails, the OSGi Service Platform must be left in the same state as it was in before the method was called.

When installing a bundle, the Framework attempts to resolve the bundle's native code dependencies. If this attempt fails, the bundle must not be installed. See *Loading Native Code Libraries* on page 53.

Once a bundle has been installed, a Bundle object is created and all remaining life-cycle operations must be performed upon this object. The returned Bundle object can be used to start, stop, update, and uninstall the bundle.

4.8.5 Resolving Bundles

A bundle can enter the RESOLVED state when the Framework has successfully resolved the bundle's code dependencies. These dependencies include:

- Classpath dependencies from the bundle's Bundle-Classpath manifest header.
- Package dependencies from the bundle's Export-Package and Import-Package manifest headers.

If the bundle's dependencies are resolved, selected packages declared in the bundle's Export-Package manifest header must be exported.

A bundle may be resolved at the Framework implementation's discretion once the bundle is installed.

4.8.6 Starting Bundles

The Bundle interface defines the start() method for starting a bundle. If this method succeeds, the bundle's state is set to ACTIVE and it remains in this state until it is stopped. The optional Start Level service influences the actual starting and stopping of bundles. See *Start Level Service Specification* on page 137.

In order to be started, a bundle must first be resolved. The Framework must attempt to resolve the bundle, if not already resolved, when trying to start the bundle. If the bundle fails to resolve, the start method must throw a BundleException.

If the bundle is resolved, the bundle must be activated by calling its Bundle Activator object, if one exists. The BundleActivator interface defines methods that the Framework invokes when it starts and stops the bundle.

To inform the OSGi environment of the fully qualified class name serving as its Bundle Activator, a bundle developer must declare a Bundle-Activator manifest header in the bundle's manifest file. The Framework must instantiate a new object of this class and cast it to a BundleActivator instance. It must then call the BundleActivator.start method to start the bundle.

The following is an example of a Bundle-Activator manifest header:

```
Bundle-Activator: com.acme.BA
```

A class acting as Bundle Activator must implement the BundleActivator interface, be declared public, and have a public default constructor so an instance of it may be created with Class.newInstance.

Supplying a Bundle Activator is optional. For example, a library bundle that only exports a number of packages usually does not need to define a Bundle Activator. A bundle providing a service should do so, however, because this is the only way for the bundle to obtain its BundleContext object and get control when started.

The BundleActivator interface defines these methods for starting and stopping a bundle:

- start(BundleContext) – This method can allocate resources that a bundle needs and start threads, and also usually registers the bundle's services. If this method does not register any services, the bundle can register the services it needs at a later time, for example in a callback, as long as it is in the ACTIVE state.
- stop(BundleContext) – This method must undo all the actions of the BundleActivator.start(BundleContext) method. However, it is unnecessary to unregister services or Framework listeners because they must be cleaned up by the Framework anyway.

4.8.7 Stopping Bundles

The Bundle interface defines the stop() method for stopping a bundle. This stops a bundle and sets the bundle's state to RESOLVED.

The BundleActivator interface defines a stop(BundleContext) method, which is invoked by the Framework to stop a bundle. This method must release any resources allocated since activation. All threads associated with the stopping bundle should be stopped immediately. The threaded code may no longer use Framework related objects (such as services and BundleContext objects) once its stop method returns.

This method may unregister services. However, if the stopped bundle had registered any services, either through its BundleActivator.start method, or while the bundle was in the ACTIVE state, the Framework must automatically unregister all registered services when the bundle is stopped.

The Framework must guarantee that if a BundleActivator.start method has executed successfully, that same BundleActivator object must be called at its BundleActivator.stop method when the bundle is deactivated. After calling the stop method, that particular BundleActivator object must never be used again.

Packages exported by a stopped bundle continue to be available to other bundles. This continued export implies that other bundles can execute code from a stopped bundle, and the designer of a bundle should assure that this is not harmful. Exporting only interfaces is one way to prevent this execution when the bundle is not started. Interfaces do not contain executable code so they cannot be executed.

4.8.8 Updating Bundles

The Bundle interface defines two methods for updating a bundle:

- update() – This method updates a bundle.
- update(InputStream) – This method updates a bundle from the specified InputStream object.

The update process supports migration from one version of a bundle to a newer, backward-compatible version, of the same bundle.

A bundle Newer, is backward compatible with another bundle, Older if:

- Newer provides at least the services provided by Older.
- Each service interface in Newer is compatible (as defined in [12] *The Java Language Specification*, Section 13.5) with its counterpart in Older.
- For any package exported by Older, Newer must export the same package, which must be compatible with its counterpart in Older.

A Framework must guarantee that only one version of a bundle's classes is available at any time. If the updated bundle had exported any packages that are used by other bundles, those packages must not be updated; their old versions must remain exported until the org.osgi.service.admin.PackageAdmin.refreshPackages method has been called or the Framework is restarted.

4.8.9 Uninstalling Bundles

The Bundle interface defines a method for uninstalling a bundle from the Framework: uninstall(). This method causes the Framework to notify other bundles that the bundle is being uninstalled, and sets the bundle's state to UNINSTALLED. The Framework must remove any resources related to the bundle that it is able to remove. This method must always uninstall the bundle from the persistent storage of the Framework.

Once this method returns, the state of the OSGi Service Platform must be the same as if the bundle had never been installed, unless the uninstalled bundle has exported any packages (via its Export-Package manifest header) and was selected by the Framework as the exporter of these packages.

If the bundle did export any packages that are used by other bundles, the Framework must continue to make these packages available to their importing bundles until one of the following conditions is satisfied:

- The org.osgi.service.admin.PackageAdmin.refreshPackages method has been called.
- The Framework is restarted.

4.9 The Bundle Context

The relationship between the Framework and its installed bundles is realized by the use of BundleContext objects. A BundleContext object represents the execution context of a single bundle within the OSGi Service Platform, and acts as a proxy to the underlying Framework.

A BundleContext object is created by the Framework when a bundle is started. The bundle can use this private BundleContext object for the following purposes:

- Installing new bundles into the OSGi environment. See *Installing Bundles* on page 58.
- Interrogating other bundles installed in the OSGi environment. See *Getting Bundle Information* on page 62.
- Obtaining a persistent storage area. See *Persistent Storage* on page 62.
- Retrieving service objects of registered services. See *ServiceReference Objects* on page 66.
- Registering services in the Framework service. See *Registering Services* on page 66.
- Subscribing or unsubscribing to events broadcast by the Framework. See *Events* on page 77.

When a bundle is started, the Framework creates a BundleContext object and provides this object as an argument to the start(BundleContext) method of the bundle's Bundle Activator. Each bundle is provided with its own BundleContext object; these objects should not be passed between bundles, as the BundleContext object is related to the security and resource allocation aspects of a bundle.

After the stop(BundleContext) method is called, the BundleContext object must no longer be used. Framework implementations must throw an exception if the BundleContext object is used after a bundle is stopped.

4.9.1 Getting Bundle Information

The BundleContext interface defines methods which can be used to retrieve information about bundles installed in the OSGi Service Platform:

- getBundle() – Returns the single Bundle object associated with the BundleContext object.
- getBundles() – Returns an array of the bundles currently installed in the Framework.
- getBundle(long) – Returns the Bundle object specified by the unique identifier, or null if no matching bundle is found.

Bundle access is not restricted; any bundle can enumerate the set of installed bundles. Information that can identify a bundle, however (such as its location, or its header information), is only provided to callers that have AdminPermission.

4.9.2 Persistent Storage

The Framework should provide a private persistent storage area for each installed bundle on platforms with some file system support.

The BundleContext interface defines access to this storage in terms of the File class, which supports platform-independent definitions of file and directory names.

The BundleContext interface defines a method to access the private persistent storage area: getDataFile(String). This method takes a relative file name as an argument and translates it into an absolute file name in the bundle's persistent storage area and returns a File object. This method returns null if there is no support for persistent storage.

The Framework must automatically provide the bundle with FilePermission[READ | WRITE | DELETE,<storage area>] to allow the bundle to read, write, and delete files in that storage area.

Further FilePermissions for this area can be set with a relative path name. For example, FilePermission[EXECUTE,bin/*] specifies that the sub-directory in the bundle's private data area may contain executables (this only provides execute permission within the Java environment and does not handle the potential underlying operating system issues related to executables).

This special treatment applies only to FilePermission objects assigned to a bundle. Default permissions must not receive this special treatment. A FilePermission for a relative path name assigned via the setDefaultPermission method must be ignored.

4.9.3 Environment Properties

The BundleContext interface defines a method for returning information pertaining to Framework properties: getProperty(String). This method can be used to return the following Framework properties:

Property name	Description
org.osgi.framework.version	The specification version of the Framework.
org.osgi.framework.vendor	The vendor of the Framework implementation.
org.osgi.framework.language	The language being used. See *ISO 639, International Standards Organization* See [16] *Codes for the Representation of Names of Languages* for valid values.
org.osgi.framework.executionenvironment	A comma separated list of provided Execution Environments (EE). All methods of each listed EE must be present on the Service Platform. For example, this property could contain: `CDC-1.0/Foundation-1.0,OSGi/Minimum-1.0` A Service Platform implementation must provide *all* the signatures that are defined in the mentioned EEs. Thus the Execution Environment for a specific Service Platform Server must be the combined set of all signatures of all EEs in the org.osgi.framework.executionenvironment property.
org.osgi.framework.processor	Processor name. The following table defines a list of processor names. New processors are made available on the OSGi web site in the Developers Zone. Names should be matched case insensitive.

Name	Aliases	Description
68k		68000 and up
ARM		Intel Strong ARM

Table 4 Property Names

Property name	Description

Alpha		Compaq
Ignite	psc1k	PTSC
Mips		SGI
PArisc		Hewlett Packard
PowerPC	power ppc	Motorola/IBM
Sparc		SUN
x86	pentium i386 i486 i586 i686	Intel

org.osgi.framework.os.version

The version of the operating system. If the version does not fit the standard x.y.z format (e.g. 2.4.32-kwt), then the Operator should define a System property with this name.

org.osgi.framework.os.name

The name of the operating system (OS) of the host computer. The following table defines a list of OS names. New OS names are made available on the OSGi web site in the Developers Zone. Names should be matched case insensitive.

Name	Aliases	Description
AIX		IBM
DigitalUnix		Compaq
FreeBSD		Free BSD
HPUX		Hewlett Packard
IRIX		Silicon Graphics
Linux		Open source
MacOS		Apple
Netware		Novell
OpenBSD		Open source
NetBSD		Open source
OS2	OS/2	IBM
QNX	procnto	QNX
Solaris		Sun Micro Systems
SunOS		Sun Micro Systems
VxWorks		WindRiver Systems
Windows95	Win95 Windows 95	Microsoft Windows 95
Windows98	Win98 Windows 98	Microsoft Windows 98

Table 4 *Property Names*

Property name		Description
WindowsNT	WinNT Windows NT	Microsoft Windows NT
WindowsCE	WinCE Windows CE	Microsoft Windows CE
Windows2000	Win2000 Windows 2000	Microsoft Windows 2000
WindowsXP	Windows XP, WinXP	Microsoft Windows XP

Table 4 *Property Names*

All Framework properties may be defined by the Operator as System properties. If these properties are not defined as System properties, the Framework must construct these properties from relevant standard Java System properties.

The alias list is names that have been reported to be returned by certain versions of the related operating systems. Frameworks should try to convert these aliases to the canonical OS or processor name. The bundle developer should use the canonical name in the Bundle-NativeCode manifest header.

4.10 Services

In the OSGi Service Platform, bundles are built around a set of cooperating services available from a shared service registry. Such an OSGi service is defined semantically by its *service interface* and implemented as a *service object*.

The service interface should be specified with as few implementation details as possible. OSGi has specified many service interfaces for common needs and will specify more in the future.

The service object is owned by, and runs within, a bundle. This bundle must register the service object with the Framework service registry so that the service's functionality is available to other bundles under control of the Framework.

Dependencies between the bundle owning the service and the bundles using it are managed by the Framework. For example, when a bundle is stopped, all the services registered with the Framework by that bundle must be automatically unregistered.

The Framework maps services to their underlying service objects, and provides a simple but powerful query mechanism that enables an installed bundle to request the services it needs. The Framework also provides an event mechanism so that bundles can receive events of service objects that are registered, modified, or unregistered.

4.10.1 ServiceReference Objects

In general, registered services are referenced through ServiceReference objects. This avoids creating unnecessary dynamic service dependencies between bundles when a bundle needs to know about a service but does not require the service object itself.

A ServiceReference object can be stored and passed on to other bundles without the implications of dependencies. When a bundle wishes to use the service, it can be obtained by passing the ServiceReference object to BundleContext.getService(ServiceReference). See *Obtaining Services* on page 70.

A ServiceReference object encapsulates the properties and other meta information about the service object it represents. This meta information can be queried by a bundle to assist in the selection of a service that best suits its needs.

When a bundle queries the Framework service registry for services, the Framework must provide the requesting bundle with the ServiceReference objects of the requested services, rather than with the services themselves.

Getting a ServiceReference object from a ServiceRegistration object must not require any permission.

A ServiceReference object is valid only as long as the service object it references has not been unregistered. However, its properties must remain available as long as the ServiceReference object exists.

4.10.2 Service Interfaces

A *service interface* is the specification of the service's public methods.

In practice, a bundle developer creates a service object by implementing its service interface and registers the service with the Framework service registry. Once a bundle has registered a service object under an interface/class name, the associated service can be acquired by bundles under that interface name, and its methods can be accessed by way of its service interface.

When requesting a service object from the Framework, a bundle can specify the name of the service interface that the requested service object must implement. In the request, the bundle may optionally specify a filter string to further narrow the search.

Many service interfaces are defined and specified by organizations such as the OSGi organization. A service interface that has been accepted as a standard can be implemented and used by any number of bundle developers.

4.10.3 Registering Services

A bundle introduces a service by registering a service object with the Framework service registry. A service object registered with the Framework is exposed to other bundles installed in the OSGi environment.

Every registered service object has a unique ServiceRegistration object, and has one or more ServiceReference objects that refer to it. These ServiceReference objects expose the registration properties of the service object, including the set of service interfaces/classes they implement. The ServiceReference object can then be used to acquire a service object that implements the desired service interface.

The Framework permits bundles to register and unregister service objects dynamically. Therefore, a bundle is permitted to register service objects from the time its BundleActivator.start method is called until its BundleActivator.stop method is called and returns.

A bundle registers a service object with the Framework by calling one of the BundleContext.registerService methods on its BundleContext object:

- registerService(String,Object,Dictionary) – For a service object registered under a single service interface of which it is an instance.
- registerService(String[],Object,Dictionary) – For a service object registered under multiple service interfaces of which it is an instance.

The names of the service interfaces under which a bundle wants to register its service are provided as arguments to the BundleContext.registerService method. The Framework must ensure that the service object actually is an instance of all the service interfaces specified by the arguments, except for a Service Factory. See *Service Factories* on page 74.

To perform this check, the Framework must load the Class object for each specified service interface from either the bundle or a shared package. See *Sharing Packages* on page 46. For each Class object, Class.isInstance must be called and return true on the Class object with the service object as the argument.

The service object being registered may be further described by a Dictionary object, which contains the properties of the service as a collection of key/value pairs.

The service interface names under which a service object has been successfully registered are automatically added to the service object's properties under the key objectClass. This value must be set automatically by the Framework and any value provided by the bundle must be overridden.

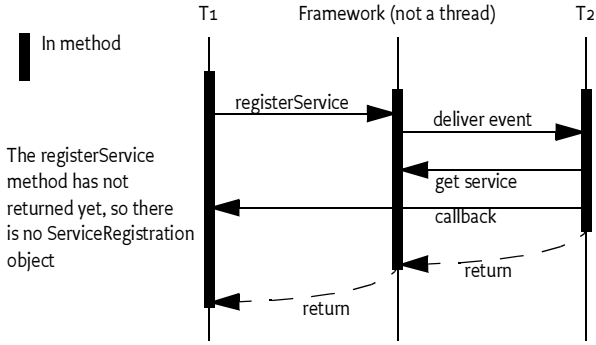
If the service object is successfully registered, the Framework must return a ServiceRegistration object to the caller. A service object can be unregistered only by the holder of its ServiceRegistration object (see the unregister() method). Every successful service object registration must yield a unique ServiceRegistration object even if the same service object is registered multiple times.

Using the ServiceRegistration object is the only way to reliably change the service object's properties after it has been registered (see setProperties(Dictionary)). Modifying a service object's Dictionary object after the service object is registered may not have any effect on the service's properties.

4.10.4 Early Need For ServiceRegistration Object

The registration of a service object will cause all registered ServiceListener objects to be notified. This is a synchronous notification. This means that such a listener can get access to the service and call its methods before the registerService method has returned the ServiceRegistration object. In certain cases, access to the ServiceRegistration object is necessary in such a callback. However, the registering bundle has not yet received the ServiceRegistration object. Figure 19 on page 68 shows such a sequence.

Figure 19 *Callback sequence event registration.*



In a case as described previously, access to the registration object can be obtained with a ServiceFactory object. If a ServiceFactory object is registered, the Framework must call-back the registering bundle with the ServiceFactory method getService(Bundle,ServiceRegistration). The required ServiceRegistration object is a parameter in this method.

4.10.5 Service Registration Properties

Properties hold information as key/value pairs. The key is a String object and the value should be a type recognized by Filter objects (see *Filters* on page 73 for a list). Multiple values for the same key are supported with arrays ([]) and Vector objects.

The values of properties should be limited to primitive or standard Java types to prevent unwanted interbundle dependencies. The Framework cannot detect dependencies that are created by the exchange of objects between bundles via the service properties.

The key of a property is not case sensitive. ObjectClass, OBJECTCLASS and objectclass all are the same property key. A Framework must, however, return the key in ServiceReference.getPropertyKeys in exactly the same case as it was last set. When a Dictionary object that contains keys that only differ in case is passed, the Framework must raise an exception.

The properties of a ServiceRegistration object are intended to provide information *about* the service object. The properties should not be used to participate in the actual function of the service. Modifying the properties for the service registration is a potentially expensive operation. For example, a Framework may pre-process the properties into an index during registration to speed up later look-ups.

The Filter interface supports complex filtering and can be used to find matching service objects. Therefore, all properties share a single name-space in the Framework service registry. As a result, it is important to use descriptive names or formal definitions of shorter names to prevent conflicts. Several OSGi specifications reserve parts of this name-space. All properties starting with service. and the property objectClass are reserved for use by OSGi specifications.

Table 5 Standard Framework Service Registry Properties contains a list of predefined properties.

Property Key	Type	Constants	Property Description
objectClass	String[ ]	OBJECTCLASS	The objectClass property contains the set of class and interface names under which a service object is registered with the Framework. The Framework must set this property automatically. The Framework must guarantee that when a service object is retrieved with BundleContext.getService(ServiceReference), it can be cast to any of these classes or interfaces.
service.description	String	SERVICE_DESCRIPTION	The service.description property is intended to be used as documentation and is optional. Frameworks and bundles can use this property to provide a short description of a registered service object. The purpose is mainly for debugging because there is no support for localization.
service.id	Long	SERVICE_ID	Every registered service object is assigned a unique service.id by the Framework. This number is added to the service object's properties. The Framework assigns a unique value to every registered service object that is larger than values provided to all previously registered service objects.

Table 5 *Standard Framework Service Registry Properties*

Property Key	Type	Constants	Property Description
service.pid	String	SERVICE_PID	The service.pid property optionally identifies a persistent, unique name for the service object. This name must be assigned by the bundle registering the service and should be a unique string. Every time this service object is registered, including after a restart of the Framework, this service object should be registered with the same service.pid property value. The value can be used by other bundles to persistently store information about this service object.
service.ranking	Integer	SERVICE_RANKING	When registering a service object, a bundle may optionally specify a service.ranking number as one of the service object's properties. If multiple qualifying service interfaces exist, a service with the highest SERVICE_RANKING number, or when equal to the lowest SERVICE_ID, determines which service object is returned by the Framework.
service.vendor	String	SERVICE_VENDOR	This optional property can be used by the bundle registering the service object to indicate the vendor.

Table 5 *Standard Framework Service Registry Properties*

4.10.6 Permission Check

The process of registering a service object is subject to a permission check. The registering bundle must have ServicePermission[REGISTER,<interface name>] to register the service object under all the service interfaces specified.

Otherwise, the service object must not be registered, and a SecurityException must be thrown. See *Permission Types* on page 82 for more information.

4.10.7 Obtaining Services

In order to use a service object and call its methods, a bundle must first obtain a ServiceReference object. The BundleContext interface defines two methods a bundle can call to obtain ServiceReference objects from the Framework:

- getServiceReference(String) – This method returns a ServiceReference object to a service object that implements, and was registered under, the name of the service interface specified as String. If multiple such service objects exist, the service object with the highest SERVICE_RANKING is returned. If there is a tie in ranking, the service object with the lowest SERVICE_ID (the service object that was registered first) is returned.
- getServiceReferences(String,String) – This method returns an array of ServiceReference objects that:
  - Implement and were registered under the given service interface.
  - Satisfy the search filter specified. The filter syntax is further explained in *Filters* on page 73.

Both methods must return null if no matching service objects are returned. Otherwise the caller receives one or more ServiceReference objects. These objects can be used to retrieve properties of the underlying service object, or they can be used to obtain the actual service object via the BundleContext object.

4.10.8 Getting Service Properties

To allow for interrogation of service objects, the ServiceReference interface defines these two methods:

- getPropertyKeys() – Returns an array of the property keys that are available.
- getProperty(String) – Returns the value of a property.

Both of these methods must continue to provide information about the referenced service object, even after it has been unregistered from the Framework. This requirement can be useful when a ServiceReference object is stored with the Log Service.

4.10.9 Getting Service Objects

The BundleContext object is used to obtain the actual service object so that the Framework can account for the dependencies. If a bundle retrieves a service object, that bundle becomes dependent upon the life-cycle of that registered service object. This dependency is tracked by the BundleContext object used to obtain the service object, and is one reason that it is important to be careful when sharing BundleContext objects with other bundles.

The method BundleContext.getService(ServiceReference) returns an object that implements the interfaces as defined by the objectClass property.

This method has the following characteristics:

- Returns null if the underlying service object has been unregistered.
- Determines if the caller has ServicePermission[GET,<interface name>], to get the service object using at least one of the service interfaces under which the service was registered. This permission check is necessary so that ServiceReference objects can be passed around freely without compromising security.
- Increments the usage count of the service object by one for this BundleContext object.

- If the service object implements the ServiceFactory interface, it is not returned. Instead, if the bundle context's usage count of the service object is one, the object is cast to a ServiceFactory object and the getService method is called to create a customized service object for the calling bundle. Otherwise, a cached copy of this customized object is returned. See *Service Factories* on page 74 for more information about ServiceFactory objects.

Both of the BundleContext.getServiceReference methods require that the caller has the required ServicePermission[GET,<name>] to get the service object for the specified service interface names. If the caller lacks the required permission, these methods must return null.

4.10.10 Information About Registered Services

The Bundle interface defines these two methods for returning information pertaining to service usage of the bundles:

- getRegisteredServices() – Returns the service objects that the bundle has registered with the Framework.
- getServicesInUse() – Returns the service objects that the bundle is using.

4.11 Stale References

The Framework must manage the dependencies between bundles. This management is, however, restricted to Framework structures. Bundles must listen to events generated by the Framework to clean up and remove *stale references*.

A stale reference is a reference to a Java object that belongs to the classloader of a bundle that is stopped or is associated with a service object that is unregistered. Standard Java does not provide any generic means to clean up stale references, and bundle developers must analyze their code carefully to ensure that stale references are deleted.

Stale references are potentially harmful because they hinder the Java garbage collector from harvesting the classes, and possibly the instances, of stopped bundles. This may result in significantly increased memory usage and can cause updating native code libraries to fail. Bundles tracking services are strongly recommended to use the Service Tracker. See *Service Tracker Specification* on page 391.

Service developers can minimize the consequences (but not completely prevent) of stale references by using the following mechanisms:

- Implement service objects using the ServiceFactory interface. The methods in the ServiceFactory interface simplify tracking bundles that use their service objects. See *Service Factories* on page 74.
- Use indirection in the service object implementations. Service objects handed out to other bundles should use a pointer to the actual service object implementation. When the service object becomes invalid, the pointer is set to null, effectively removing the reference to the actual service object.

The behavior of a service that becomes unregistered is undefined. Such services may continue to work properly or throw an exception at their discretion. This type of error should be logged.

4.12 Filters

The Framework provides a Filter interface, and uses a search filter syntax in the getServiceReference(s) method that is based on an RFC 1960-based search filter string. See [13] *A String Representation of LDAP Search Filters*. Filter objects can be created by calling BundleContext.createFilter(String) with the chosen filter string.

The syntax of a filter string is based upon the string representation of LDAP search filters as defined in [13] *A String Representation of LDAP Search Filters*. It should be noted that RFC 2254: A String Representation of LDAP Search Filters supersedes RFC 1960 but only adds extensible matching and is not applicable for this OSGi Framework API.

The string representation of an LDAP search filter uses a prefix format, and is defined with the following grammar:

```
filter       ::=  '(' filter-comp ')'
filter-comp  ::=  and | or | not | item
and          ::=  '&' filter-list
or           ::=  '|' filter-list
not          ::=  '!' filter
filter-list  ::=  filter | filter filter-list
item         ::=  simple | present | substring
simple       ::=  attr filter-type value
filter-type  ::=  equal | approx | greater | less
equal        ::=  '='
approx       ::=  '~='
greater      ::=  '>='
less         ::=  '<='
present      ::=  attr '=*'
substring    ::=  attr '=' initial any final
inital       ::=  () | value
any          ::=  '*' star-value
star-value   ::=  () | value '*' star-value
final        ::=  () | value
```

attr is a string representing an attribute, or key, in the properties objects of the services registered with the Framework. Attribute names are not case sensitive; that is, cn and CN both refer to the same attribute. attr must not contain the characters '=', '>', '<', '~', '(' or ')'. attr may contain embedded spaces but leading and trailing spaces must be ignored.

value is a string representing the value, or part of one, of a key in the properties objects of the registered services. If a value must contain one of the characters '*', '(' or ')', then these characters should be preceded with the backslash '(\') character. Spaces are significant in value. Space characters are defined by Character.isWhiteSpace().

Although both the substring and present productions can produce the attr=* construct, this construct is used only to denote a presence filter.

The approximate match ('~=') production is implementation specific but should at least ignore case and white space differences. Codes such as soundex or other smart *closeness* comparisons are optional.

Comparison of values is not straightforward. Strings are compared differently than numbers and it is possible for an attr to have multiple values. Keys in the match argument must always be String objects.

The comparison is defined by the object type of the attr's value. The following rules apply for comparison:

- *String objects* – String comparison
- *Integer, Long, Float, Double, Byte, Short objects* – Numerical comparison
- *Comparable objects* – Comparison through the Comparable interface,.
- *Character object* – Character class based comparison
- *Boolean objects* – Equality comparisons only
- *Array [ ] objects* – Rules are recursively applied to values
- *Vector* – Rules are recursively applied to elements

Arrays of primitives are also supported, as well as null values and mixed types. BigInteger and BigDecimal classes are not part of the minimal execution environment and should not be used when portability is an issue. The framework must use the Comparable interface to compare objects not listed.

An object that implements the Comparable interface can not compare directly with the value from the filter (a string) because the Comparable interface requires equal types. Such an object should therefore have a constructor that takes a String object as argument. If no such constructor exist, the Framework is not able to compare the object and the expression will therefore not match. Otherwise, a new object must be constructed with the value from the filter. Both the original and constructed objects can then be cast to Comparable and compared.

A Filter object can be used numerous times to determine if the match argument, a ServiceReference or a Dictionary object, matches the filter string that was used to create the Filter object.

A filter matches a key that has multiple values if it matches at least one of those values. For example,

```
Dictionary dict = new Hashtable();
dict.put( "cn", new String[] { "a", "b", "c" } );
```

The dict will match true against a filter with "(cn=a)" but also "(cn=b)".

The Filter.toString method must always return the filter string with unnecessary white space removed.

4.13 Service Factories

A Service Factory allows customization of the service object that is returned when a bundle calls BundleContext.getService(ServiceReference).

Normally, the service object that is registered by a bundle is returned directly. If, however, the service object that is registered implements the ServiceFactory interface, the Framework must call methods on this object to create a unique service object for each distinct bundle that gets the service.

When the service object is no longer used by a bundle – for example, when that bundle is stopped – then the Framework must notify the ServiceFactory object.

ServiceFactory objects help manage bundle dependencies that are not explicitly managed by the Framework. By binding a returned service object to the requesting bundle, the service object can listen to events related to that bundle and remove objects, for example listeners, registered by that bundle when it is stopped. With a Service Factory, listening to events is not even necessary, because the Framework must inform the ServiceFactory object when a service object is released by a bundle, which happens automatically when a bundle is stopped.

The ServiceFactory interface defines the following methods:

- getService(Bundle,ServiceRegistration) – This method is called by the Framework if a call is made to BundleContext.getService and the following are true:
  - The specified ServiceReference argument points to a service object that implements the ServiceFactory interface.
  - The bundle's usage count of that service object is zero; that is, the bundle currently does not have any dependencies on the service object.

  The call to BundleContext.getService must be routed by the Framework to this method, passing to it the Bundle object of the caller. The Framework must cache the mapping of the requesting bundle-to-service, and return the cached service object to the bundle on future calls to BundleContext.getService, as long as the requesting bundle's usage count of the service object is greater than zero.
  The Framework must check the service object returned by this method. If it is not an instance of all the classes named when the service factory was registered, null is returned to the caller that called getService. This check must be done as specified in *Registering Services* on page 66.
- ungetService(Bundle,ServiceRegistration,Object) – This method is called by the Framework if a call is made to BundleContext.ungetService and the following are true:
  - The specified ServiceReference argument points to a service object that implements the ServiceFactory interface.
  - The bundle's usage count for that service object must drop to zero after this call returns; that is, the bundle is about to release its last dependency on the service object.

  The call to BundleContext.ungetService must be routed by the Framework to this method so the ServiceFactory object can release the service object previously created.
  Additionally, the cached copy of the previously created service object must be unreferenced by the Framework so it may be garbage collected.

4.14 Importing and Exporting Services

The Export-Service manifest header declares the interfaces that a bundle may register. It provides advisory information that is not used by the Framework. This header is intended for use by server-side management tools.

The Export-Service manifest header must conform to the following syntax:

```
Export-Service  ::=  class-name ( ',' class-name )*
class-name      ::=  <fully qualified class name>
```

The Import-Service manifest header declares the interfaces the bundle may use. It provides advisory information that is not used by the Framework. This header is also intended for use by server-side management tools.

The Import-Service manifest header must conform to the following syntax:

```
Import-Service  ::=  class-name ( ',' class-name )*
class-name      ::= <fully qualified class name>
```

4.15 Releasing Services

In order for a bundle to release a service object, it must remove the dynamic dependency on the bundle that registered the service object. The BundleContext interface defines a method to release service objects: ungetService(ServiceReference). A ServiceReference object is passed as the argument of this method.

This method returns a boolean value:

- false if the bundle's usage count of the service object is already zero when the method was called, or the service object has already been unregistered.
- true if the bundle's usage count of the service object was more than zero before this method was called.

4.16 Unregistering Services

The ServiceRegistration interface defines the unregister() method to unregister the service object. This must remove the service object from the Framework service registry. The ServiceReference object for this ServiceRegistration object can no longer be used to access the service object.

The fact that this method is on the ServiceRegistration object ensures that only the bundle holding this object can unregister the associated service object. The bundle that unregisters a service object, however, might not be the same bundle that registered it. As an example, the registering bundle could have passed the ServiceRegistration object to another bundle, endowing that bundle with the responsibility of unregistering the service object. Passing ServiceRegistration objects should be done with caution.

After ServiceRegistration.unregister successfully completes, the service object must be:

- Completely removed from the Framework service registry. As a consequence, ServiceReference objects obtained for that service object can no longer be used to access the service object. Calling BundleContext.getService method with the ServiceReference object must return null.
- Unregistered, even if other bundles had dependencies upon it. Bundles must be notified of the unregistration through the publishing of a ServiceEvent object of type ServiceEvent.UNREGISTERING. This event is sent synchronously in order to give bundles the opportunity to release the service object.
  After receiving an event of type ServiceEvent.UNREGISTERING the bundle should release the service object and release any references it has to this object, so that the service object can be garbage collected by the Java VM.
- Released by all using bundles. For each bundle whose usage count for the service object remains greater than zero after all invoked ServiceListener objects have returned, the Framework must set the usage count to zero and release the service object.

4.17 Configurable Services

The Configurable interface is a minimalistic approach to configuration management. The Configurable service is therefore intended to be superseded by the Configuration Admin service. See *Configuration Admin Service Specification* on page 181.

A Configurable service is one that can be configured dynamically at runtime to change its behavior. As an example, a configurable Http Service may support an option to set the port number.

A service object is administered as configurable by implementing the Configurable interface, which has one method: getConfigurationObject(). This method returns an Object instance that holds the configuration data of the service. As an example, a configuration object could be implemented as a Java Bean.

The configuration object handles all the configuration aspects of a service so that the service object itself does not have to expose its configuration properties.

Before returning the configuration object, getConfigurationObject should check that the caller has the required permission to access and manipulate it, and if not, it should throw a SecurityException. Note that the required permission is implementation-dependent.

4.18 Events

The OSGi Framework supports the following types of events:

- ServiceEvent – Reports registration, unregistration, and property changes for service objects. All events of this kind must be delivered synchronously.

- BundleEvent – Reports changes in the life-cycle of bundles.
- FrameworkEvent – Reports that the Framework is started, startlevel has changed, packages have been refreshed, or that an error has been encountered.

4.18.1 Listeners

A listener interface is associated with each type of event. The following list describes these listeners.

- ServiceListener – Called with an event of type ServiceEvent when a service object has been registered or modified, or is in the process of unregistering. A security check must be performed for each registered listener when a ServiceEvent occurs. The listener must not be called unless it has the required ServicePermission[GET,<interface name>] for at least one of the interfaces under which the service object is registered.
- BundleListener and SynchronousBundleListener – Called with an event of type BundleEvent when a bundle has been installed, started, stopped, updated, or uninstalled. SynchronousBundleListener objects are called synchronously during the processing of the event, and must be called before any BundleListener object is called.
- FrameworkListener – Called with an event of type FrameworkEvent.

BundleContext interface methods are defined which can be used to add and remove each type of listener.

A bundle that uses a service object should register a ServiceListener object to track the availability of the service object, and take appropriate action when the service object is unregistering (this can be significantly simplified with the *Service Tracker Specification* on page 391).

Events can be asynchronously delivered, unless otherwise stated, meaning that they are not necessarily delivered by the same thread that generated the event. The thread used to call an event listener is not defined.

4.18.2 Delivering Events

When delivering an event asynchronously, the Framework must:

- Collect a snapshot of the listener list at the time the event is published (rather than doing so in the future just prior to event delivery) but before the event is delivered, so that listeners do not enter the list after the event happened.
- Ensure that listeners on the list at the time the snapshot is taken still belong to active bundles at the time the event is delivered.

If the Framework did not capture the current listener list when the event was published, but instead waited until just prior to event delivery, then it would be possible for a bundle to have started and registered a listener, and the bundle could see its own BundleEvent.INSTALLED event, which would be an error.

The following three scenarios illustrate this concept.

1. Scenario 1 event sequence:
   - Event A is published.
   - Listener 1 is registered.

- Asynchronous delivery of Event A is attempted.

Expected Behavior: Listener 1 must not receive Event A, because it was not registered at the time the event was published.

2. Scenario 2 event sequence:
   - Listener 2 is registered.
   - Event B is published.
   - Listener 2 is unregistered.
   - Asynchronous delivery of Event B is attempted.

   Expected Behavior: Listener 2 receives Event B, because Listener 2 was registered at the time Event B was published.
3. Scenario 3 event sequence:
   - Listener 3 is registered.
   - Event C is published.
   - The bundle that registered Listener 3 is stopped.
   - Asynchronous delivery of Event C is attempted.

   Expected Behavior: Listener 3 must not receive Event C, because its Bundle Context object is invalid.

4.18.3 Synchronization Pitfalls

As a general rule, a Java monitor should not be held when event listeners are called. This means that neither the Framework nor the originator of a synchronous event should be in a monitor when a callback is initiated.

The purpose of a Java monitor is to protect the update of data structures. This should be a small region of code that does not call any code the effect of which cannot be overseen. Calling the OSGi Framework from synchronized code can cause unexpected side effects. One of these side effects might be *deadlock*. A deadlock is the situation where two threads are blocked because they are waiting for each other.

Time-outs can be used to break deadlocks, but Java monitors do not have time-outs. Therefore, the code will hang forever until the system is reset (Java has deprecated all methods that can stop a thread). This type of deadlock is prevented by not calling the Framework (or other code that might cause callbacks) in a synchronized block.

If locks are necessary when calling other code, use the Java monitor to create semaphores that can time-out and thus provide an opportunity to escape a deadlocked situation.

4.19 Framework Startup and Shutdown

A Framework implementation must be started before any services can be provided. The details of how a Framework should be started is not defined in this specification, and may be different for different implementations. Some Framework implementations may provide command line options, and others may read startup information from a configuration file. In all cases, Framework implementations must perform all of the following actions in the given order.

4.19.1 Startup

When the Framework is started, the following actions must occur:

1. Event handling is enabled. Events can now be delivered to listeners. Events are discussed in *Events* on page 77.
2. The system bundle enters the STARTING state. More information about the system bundle can be found in *The System Bundle* on page 42.
3. A bundle's state is persistently recorded in the OSGi environment. When the Framework is restarted, all installed bundles previously recorded as being started must be started as described in the Bundle.start method. Any exceptions that occur during startup must be published as a Framework event of type FrameworkEvent.ERROR. Bundles and their different states are discussed in *The Bundle Object* on page 57. If the Framework implements the optional Start Level specification, this behavior is different. See *Start Level Service Specification* on page 137.
4. The system bundle enters the ACTIVE state.
5. A Framework event of type FrameworkEvent.STARTED is broadcast.

4.19.2 Shutdown

The Framework will also need to be shut down on occasion. Shutdown can also be initiated by stopping the system bundle, covered in *The System Bundle* on page 42. When the Framework is shut down, the following actions must occur in the given order:

1. The system bundle enters the STOPPING state.
2. All ACTIVE bundles are suspended as described in the Bundle.stop method, except that their persistently recorded state indicates that they must be restarted when the Framework is next started. Any exceptions that occur during shutdown must be published as a Framework event of type FrameworkEvent.ERROR. If the Framework implements the optional Start Level specification, this behavior is different. See *Start Level Service Specification* on page 137.
3. Event handling is disabled.

4.20 Security

The Framework security model is based on the Java 2 specification. If security checks are performed, they must be done according to [14] *The Java Security Architecture for JDK 1.2*. It is assumed that the reader is familiar with this specification.

The Java platform on which the Framework runs must provide the Java Security APIs necessary for Java 2 permissions. On resource-constrained platforms, these Java Security APIs may be stubs that allow the bundle classes to be loaded and executed, but the stubs never actually perform the security checks. The behavior of these stubs must be as follows:

- *checkPermission* – Return without throwing a SecurityException.
- *checkGuard* – Return without throwing a SecurityException.

- *implies* – Return true.

This behavior allows code to run as if all bundles have AllPermission.

Many of the Framework methods require the caller to explicitly have certain permissions when security is enabled. Services may also have permissions specific to them that provide more finely grained control over the operations that they are allowed to perform. Thus a bundle that exposes service objects to other bundles may also need to define permissions specific to the exposed service objects.

For example, the User Admin service has an associated UserAdminPermission class that is used to control access to this service.

4.20.1 Permission Checks

When a permission check is done, java.security.AccessController should check all the classes on the call stack to ensure that every one of them has the permission being checked.

Because service object methods often allow access to resources to which only the bundle providing the service object normally has access, a common programming pattern uses java.security.AccessController.doPrivileged in the implementation of a service object. The service object can assume that the caller is authorized to call the service object because a service can only be obtained with the appropriate ServicePermission[GET,<interface name>]. It should therefore use only its own permissions when it performs its function.

As an example, the dial method of a fictitious PPP Service accesses the serial port to dial a remote server and start up the PPP daemon. The bundle providing the PPP Service will have permission to execute programs and access the serial port, but the bundles using the PPP Service may not have those permissions.

When the dial method is called, the first check will be to ensure that the caller has permission to dial. This check is done with the following code:

```
SecurityManager sm = System.getSecurityManager();
if ( sm != null )
   sm.checkPermission( new com.acme.ppp.DialPermission() );
```

If the permission check does not throw an exception, the dial method must now enter a privileged state to actually cause the modem to dial and start the PPP daemon as shown in the following example.

```
Process proc = (Process)
   AccessController.doPrivileged(newPrivilegedAction() {
        public Object run() {
           Process proc = null;
           if ( connectToServer() )
              proc = startDaemon();
           return proc;
        }
     }
   );
```

For alternate ways of executing privileged code, see [14] *The Java Security Architecture for JDK 1.2*.

4.20.2 Privileged Callbacks

The following interfaces define bundle callbacks that are invoked by the Framework:

- BundleActivator
- ServiceFactory
- Bundle-, Service-, and FrameworkListener.

When any of these callbacks are invoked by the Framework, the bundle that caused the callback may still be on the stack. For example, when one bundle installs and then starts another bundle, the installer bundle may be on the stack when the BundleActivator.start method of the installed bundle is called. Likewise, when a bundle registers a service object, it may be on the stack when the Framework calls back the serviceChanged method of all qualifying ServiceListener objects.

Whenever any of these bundle callbacks try to access a protected resource or operation, the access control mechanism should consider not only the permissions of the bundle receiving the callback, but also those of the Framework and any other bundles on the stack. This means that in these callbacks, bundle programmers normally would use doPrivileged calls around any methods protected by a permission check (such as getting or registering service objects).

In order to reduce the number of doPrivileged calls by bundle programmers, the Framework must perform a doPrivileged call around any bundle callbacks. The Framework should have java.security.AllPermission. Therefore, a bundle programmer can assume that the bundle is not further restricted except for its own permissions.

Bundle programmers do not need to use doPrivileged calls in their implementations of any callbacks registered with and invoked by the Framework.

For any other callbacks that are registered with a service object and therefore get invoked by the service-providing bundle directly, doPrivileged calls must be used in the callback implementation if the bundle's own privileges are to be exercised. Otherwise, the callback must fail if the bundle that initiated the callback lacks the required permissions.

A framework must never load classes in a doPrivileged region, but must instead use the current stack. This means that static initializers should never assume that they are privileged. Any privileged code in a static initializer must be guarded with a doPrivileged region in the static initializer.

4.20.3 Permission Types

The following permission types are defined by the Framework:

- AdminPermission – Enables access to the administrative functions of the Framework.
- ServicePermission – Controls service object registration and access.
- PackagePermission – Controls importing and exporting packages.

4.20.4 AdminPermission

An AdminPermission has no parameters associated with it and is always named admin. AdminPermission is required by all administrative functions. AdminPermission has no actions.

4.20.5 Service Permission

A ServicePermission has the following parameters.

- *Interface Name* – The interface name may end with a wildcard to match multiple interface names. (See java.security.BasicPermission for a discussion of wildcards.)
- *Action* – Supported actions are: REGISTER – Indicates that the permission holder may register the service object, and GET – Indicates that the holder may get the service.

When an object is being registered as a service object using BundleContext.registerService, the registering bundle must have the ServicePermission to register all the named classes. See *Registering Services* on page 66.

When a ServiceReference object is obtained from the service registry using BundleContext.getServiceReference or BundleContext.getServiceReferences, the calling bundle must have the required ServicePermission[GET,<interface name>] to get the service object with the named class. See *ServiceReference Objects* on page 66.

When a service object is obtained from a ServiceReference object using BundleContext.getService(ServiceReference), the calling code must have the required ServicePermission[GET,<name>] to get the service object for at least one of the classes under which it was registered.

ServicePermission must be used as a filter for the service events received by the Service Listener, as well as for the methods to enumerate services, including Bundle.getRegisteredServices and Bundle.getServicesInUse. The Framework must assure that a bundle must not be able to detect the presence of a service that it does not have permission to access.

4.20.6 Package Permission

Bundles can only import and export packages for which they have the required permission actions. A PackagePermission must be valid across all versions of a package.

A PackagePermission has two parameters:

- The package that may be exported. A wildcard may be used. The granularity of the permission is the package, not the class name.
- The action, either IMPORT or EXPORT. If a bundle has permission to export a package, the Framework must automatically grant it permission to import the package.

A PackagePermission with * and EXPORT as parameters would be able to import and export any package.

4.20.7 Bundle Permissions

The Bundle interface defines a method for returning information pertaining to a bundle's permissions: hasPermission(Object). This method returns true if the bundle's Protection Domain has the specified permission, and false if it does not or if the object specified by the argument is not an instance of java.security.Permission.

The parameter type is Object so that the Framework can be implemented on Java platforms that do not support Java 2 based security.

4.21 The Framework on Java 1.1

The Framework specification was authored assuming a Java 2 based run-time environment. This section addresses issues in implementing and deploying the OSGi Framework on Java 1.1 based run-time environments.

Overall, the OSGi specifications strive to allow implementations on Java 1.1 by not using classes in the APIs that are not available on Java 1. For example, none of the APIs use Permission classes or classes of the collection framework. However, some specified semantics can only be implemented in a Java 2 environment.

4.21.1 ClassLoader.getResource

In JDK 1.1, the ClassLoader class does not provide the findResource method. Therefore, references to the findResource method in this document refer to the getResource method.

4.21.2 ClassLoader.findLibrary

Java 2 introduced the findLibrary method, which allows classloaders to participate in the loading of native libraries. In JDK 1.1, all native code libraries must be available on a single, global library path. Therefore, native code libraries from different bundles have to reside in the same directory. If libraries have the same name, unresolvable conflicts may occur.

4.21.3 Resource URL

A bundle's classloader returns resource URL objects which use a Framework implementation-specific URLStreamHandler sub-class to capture security information about the caller.

Prior to Java 2 no constructor which took a URLStreamHandler object argument existed, requiring the Framework implementation to register a URLStreamHandler object. Conceivably, then, other code than the Framework implementation could create this type of URL object with falsified security information, and is thus a security threat. Therefore, the Bundle.getResource method cannot be implemented securely in Java versions prior to Java 2.

4.21.4 Comparable

The Filter is defined using the Comparable interface. This interface was introduced in Java 2. Framework implementations that run on Java 1.1 must take special care that this interface will not be available. The actual check should therefore take place in code that can detect the presence of this interface without linking to it, for example, using reflection.

4.22 Changes

This section defines the changes since the OSGi Service Platform Release 2.

4.22.1 Dynamic Import

A new bundle manifest header is added, DynamicImport-Package, that allows a bundle to import packages of which it has no a priori knowledge. Dynamic import is described in *Dynamically Importing Packages* on page 48.

4.22.2 Automatic Import of Java

A Framework must implicitly import all packages starting with java. for each bundle. This is discussed in *Automatically Importing java.** on page 56.

4.22.3 Native Code

The native code selection algorithm in Service Platform Release 2 selected a lesser matching operating system when certain language dependent libraries were present. The algorithm has been replaced with a more declarative description. See *Native Code Algorithm* on page 55. A common mistake with the native code clause was also highlighted with an example.

4.22.4 Synchronization Pitfalls

Using synchronized with Framework callbacks can cause deadlocks. A section has been added that discusses these issues. See *Synchronization Pitfalls* on page 79.

4.22.5 New Constants

A number of new constants have been added to the Constants class.

4.22.6 Different Default File Permissions

The interpretation of a relative file name in a FilePermission object was changed from undefined to relative to the bundle's private persistent storage area. This is discussed in *Persistent Storage* on page 62.

4.22.7 Use of System Properties

The mapping of System properties to the BundleContext.getProperty method has been clarified in *Environment Properties* on page 63.

4.22.8 Registering Services Under Classes of Non Imported Packages

When a service object is registered with the Framework, a number of interfaces/classes are specified under which the service should be registered. Previously, it was not specified from where those classes originated, and some Framework implementations decided to limit the source of these classes to be from imported or private packages. This made it impossible to register an object that was obtained from another bundle. This specification explicitly allows services to be registered under class names that are not available to the bundle.

4.22.9 Removed Reference to BigInteger/BigDecimal

The specification of the Filter class was the only reference to the BigInteger and BigDecimal classes. These classes are not part of the minimal execution requirements. These classes implement the Comparable interface so the Filter is now required to support this interface and the classes are no longer required.

4.22.10 Security

The behavior of the Permission related stub classes in the OSGi Framework have been defined and the security state of static initializers has been clarified. See *Security* on page 80.

4.22.11 Bundle-RequiredExcecutionEnvironment

A new manifest header is introduced to specify a bundle's requirements on the execution environment. This is explained in *Execution Environment* on page 52.

4.22.12 Filter name allows spaces

It was not well specified if the <attr> in the Filter syntax description allowed spaces or not. This has been clarified to allow spaces. See *Filters* on page 73 for more information.

4.22.13 Source of FrameworkEvent.STARTED

The FrameworkEvent class was updated to use the System Bundle for the STARTED event rather than null. The two argument constructor is deprecated.

4.22.14 Early Access to ServiceRegistration

A section was added describing how the ServiceRegistration object can be obtained before the registerService method had returned. See *Early Need For ServiceRegistration Object* on page 68.

4.22.15 Minor clarifications

- The javadoc for BundleContext.createFilter was updated to explicitly declare that it may throw a NullPointerException.

- The javadoc for BundleContext.installBundle and Bundle.update methods that take an InputStream object argument were updated to explicitly declare that they may throw a SecurityException.
- The javadoc description in org.osgi.framework.Constants.java for the constant OBJECTCLASS was updated to state the type is String[].

4.23 org.osgi.framework

The OSGi Framework 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.framework; specification-version=1.2
```

4.23.1 Summary

- AdminPermission – Indicates the caller's authority to perform lifecycle operations on or to get sensitive information about a bundle. [p.87]
- Bundle – An installed bundle in the Framework. [p.88]
- BundleActivator – Customizes the starting and stopping of this bundle. [p.97]
- BundleContext – A bundle's execution context within the Framework. [p.98]
- BundleEvent – A Framework event describing a bundle lifecycle change. [p.108]
- BundleException – A Framework exception used to indicate that a bundle lifecycle problem occurred. [p.43]
- BundleListener – A BundleEvent listener. [p.109]
- Configurable – Supports a configuration object. [p.110]
- Constants – Defines standard names for the OSGi environment property, service property, and Manifest header attribute keys. [p.110]
- Filter – An RFC 1960-based Filter. [p.116]
- FrameworkEvent – A general Framework event. [p.117]
- FrameworkListener – A FrameworkEvent listener. [p.119]
- InvalidSyntaxException – A Framework exception. [p.119]
- PackagePermission – A bundle's authority to import or export a package. [p.120]
- ServiceEvent – A service lifecycle change event. [p.121]
- ServiceFactory – Allows services to provide customized service objects in the OSGi environment. [p.123]
- ServiceListener – A ServiceEvent listener. [p.124]
- ServicePermission – Indicates a bundle's authority to register or get a service. [p.124]
- ServiceReference – A reference to a service. [p.125]
- ServiceRegistration – A registered service. [p.127]
- SynchronousBundleListener – A synchronous BundleEvent listener. [p.128]

4.23.2 public final class AdminPermission extends BasicPermission

Indicates the caller's authority to perform lifecycle operations on or to get sensitive information about a bundle.

`AdminPermission` has no actions or target.

The `hashCode()` method of `AdminPermission` is inherited from `java.security.BasicPermission`. The hash code it returns is the hash code of the name "AdminPermission", which is always the same for all instances of `AdminPermission`.

4.23.2.1 public AdminPermission( )

- Creates a new `AdminPermission` object with its name set to "AdminPermission".

4.23.2.2 public AdminPermission(String name, String actions)

name Ignored; always set to "AdminPermission".

actions Ignored.

- Creates a new `AdminPermission` object for use by the Policy object to instantiate new `Permission` objects.

4.23.2.3 public boolean equals(Object obj)

obj The object being compared for equality with this object.

- Determines the equality of two `AdminPermission` objects.

  Two `AdminPermission` objects are always equal.

Returns `true` if `obj` is an `AdminPermission`; `false` otherwise.

4.23.2.4 public boolean implies(Permission p)

p The permission to interrogate.

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

  This method returns `true` if the specified permission is an instance of `AdminPermission`.

Returns `true` if the permission is an instance of this class; `false` otherwise.

4.23.2.5 public PermissionCollection newPermissionCollection( )

- Returns a new `PermissionCollection` object suitable for storing `AdminPermissions`.

Returns A new `PermissionCollection` object.

4.23.3 public interface Bundle

An installed bundle in the Framework.

A `Bundle` object is the access point to define the life cycle of an installed bundle. Each bundle installed in the OSGi environment will have an associated `Bundle` object.

A bundle will have a unique identity, a long, chosen by the Framework. This identity will not change during the life cycle of a bundle, even when the bundle is updated. Uninstalling and then reinstalling the bundle will create a new unique identity.

A bundle can be in one of six states:

- UNINSTALLED[p.90]
- INSTALLED[p.89]
- RESOLVED[p.89]
- STARTING[p.90]
- STOPPING[p.90]
- ACTIVE[p.89]

Values assigned to these states have no specified ordering; they represent bit values that may be ORed together to determine if a bundle is in one of the valid states.

A bundle should only execute code when its state is one of STARTING, ACTIVE, or STOPPING. An UNINSTALLED bundle can not be set to another state; it is a zombie and can only be reached because invalid references are kept somewhere.

The Framework is the only entity that is allowed to create Bundle objects, and these objects are only valid within the Framework that created them.

4.23.3.1 public static final int ACTIVE = 32

This bundle is now running.

A bundle is in the ACTIVE state when it has been successfully started.

The value of ACTIVE is 0x00000020.

4.23.3.2 public static final int INSTALLED = 2

This bundle is installed but not yet resolved.

A bundle is in the INSTALLED state when it has been installed in the Framework but cannot run.

This state is visible if the bundle's code dependencies are not resolved. The Framework may attempt to resolve an INSTALLED bundle's code dependencies and move the bundle to the RESOLVED state.

The value of INSTALLED is 0x00000002.

4.23.3.3 public static final int RESOLVED = 4

This bundle is resolved and is able to be started.

A bundle is in the RESOLVED state when the Framework has successfully resolved the bundle's dependencies. These dependencies include:

- The bundle's class path from its Constants.BUNDLE_CLASSPATH[p.110] Manifest header.
- The bundle's package dependencies from its Constants.EXPORT_PACKAGE[p.113] and Constants.IMPORT_PACKAGE[p.114] Manifest headers.

Note that the bundle is not active yet. A bundle must be put in the RESOLVED state before it can be started. The Framework may attempt to resolve a bundle at any time.

The value of RESOLVED is 0x00000004.

4.23.3.4 public static final int STARTING = 8

This bundle is in the process of starting.

A bundle is in the STARTING state when the start[p.93] method is active. A bundle will be in this state when the bundle's BundleActivator.start[p.97] is called. If this method completes without exception, then the bundle has successfully started and will move to the ACTIVE state.

The value of STARTING is 0x00000008.

4.23.3.5 public static final int STOPPING = 16

This bundle is in the process of stopping.

A bundle is in the STOPPING state when the stop[p.94] method is active. A bundle will be in this state when the bundle's BundleActivator.stop[p.98] method is called. When this method completes the bundle is stopped and will move to the RESOLVED state.

The value of STOPPING is 0x00000010.

4.23.3.6 public static final int UNINSTALLED = 1

This bundle is uninstalled and may not be used.

The UNINSTALLED state is only visible after a bundle is uninstalled; the bundle is in an unusable state and all references to the Bundle object should be released immediately.

The value of UNINSTALLED is 0x00000001.

4.23.3.7 public long getBundleId( )

□ Returns this bundle's identifier. The bundle is assigned a unique identifier by the Framework when it is installed in the OSGi environment.

A bundle's unique identifier has the following attributes:

- Is unique and persistent.
- Is a long.
- Its value is not reused for another bundle, even after the bundle is uninstalled.
- Does not change while the bundle remains installed.
- Does not change when the bundle is updated.

This method will continue to return this bundle's unique identifier while this bundle is in the UNINSTALLED state.

Returns The unique identifier of this bundle.

4.23.3.8 public Dictionary getHeaders( )

□ Returns this bundle's Manifest headers and values. This method returns all the Manifest headers and values from the main section of the bundle's Manifest file; that is, all lines prior to the first blank line.

Manifest header names are case-insensitive. The methods of the returned Dictionary object will operate on header names in a case-insensitive manner.

For example, the following Manifest headers and values are included if they are present in the Manifest file:

```
Bundle-Name
Bundle-Vendor
Bundle-Version
Bundle-Description
Bundle-DocURL
Bundle-ContactAddress
```

This method will continue to return Manifest header information while this bundle is in the UNINSTALLED state.

Returns A Dictionary object containing this bundle's Manifest headers and values.

Throws SecurityException – If the caller does not have the AdminPermission, and the Java Runtime Environment supports permissions.

4.23.3.9 public String getLocation( )

- Returns this bundle's location identifier.

  The bundle location identifier is the location passed to BundleContext.installBundle[p.104] when a bundle is installed.

  This method will continue to return this bundle's location identifier while this bundle is in the UNINSTALLED state.

Returns The string representation of this bundle's location identifier.

Throws SecurityException – If the caller does not have the appropriate AdminPermission, and the Java Runtime Environment supports permissions.

4.23.3.10 public ServiceReference[] getRegisteredServices( )

- Returns this bundle's ServiceReference list for all services it has registered or null if this bundle has no registered services.

  If the Java runtime supports permissions, a ServiceReference object to a service is included in the returned list only if the caller has the ServicePermission to get the service using at least one of the named classes the service was registered under.

  The list is valid at the time of the call to this method, however, as the Framework is a very dynamic environment, services can be modified or unregistered at anytime.

Returns An array of ServiceReference objects or null.

Throws IllegalStateException – If this bundle has been uninstalled.

See Also ServiceRegistration[p.127], ServiceReference[p.125], ServicePermission[p.124]

4.23.3.11 public URL getResource(String name)

name The name of the resource. See java.lang.ClassLoader.getResource for a description of the format of a resource name.

- Find the specified resource in this bundle. This bundle's class loader is called to search for the named resource. If this bundle's state is INSTALLED, then only this bundle will be searched for the specified resource. Imported packages cannot be searched when a bundle has not been resolved.

Returns a URL to the named resource, or null if the resource could not be found or if the caller does not have the AdminPermission, and the Java Runtime Environment supports permissions.

Throws IllegalStateException – If this bundle has been uninstalled.

Since 1.1

4.23.3.12 public ServiceReference[] getServicesInUse( )

- Returns this bundle's ServiceReference list for all services it is using or returns null if this bundle is not using any services. A bundle is considered to be using a service if its use count for that service is greater than zero.

  If the Java Runtime Environment supports permissions, a ServiceReference object to a service is included in the returned list only if the caller has the ServicePermission to get the service using at least one of the named classes the service was registered under.

  The list is valid at the time of the call to this method, however, as the Framework is a very dynamic environment, services can be modified or unregistered at anytime.

Returns An array of ServiceReference objects or null.

Throws IllegalStateException – If this bundle has been uninstalled.

See Also ServiceReference[p.125], ServicePermission[p.124]

4.23.3.13 public int getState( )

- Returns this bundle's current state.

  A bundle can be in only one state at any time.

Returns An element of UNINSTALLED, INSTALLED, RESOLVED, STARTING, STOPPING, ACTIVE.

4.23.3.14 public boolean hasPermission(Object permission)

permission The permission to verify.

- Determines if this bundle has the specified permissions.

  If the Java Runtime Environment does not support permissions, this method always returns true.

  permission is of type Object to avoid referencing the java.security.Permission class directly. This is to allow the Framework to be implemented in Java environments which do not support permissions.

  If the Java Runtime Environment does support permissions, this bundle and all its resources including nested JAR files, belong to the same java.security.ProtectionDomain; that is, they will share the same set of permissions.

Returns true if this bundle has the specified permission or the permissions possessed by this bundle imply the specified permission; false if this bundle does not

have the specified permission or `permission` is not an `instanceof java.security.Permission`.

Throws `IllegalStateException` – If this bundle has been uninstalled.

4.23.3.15 public void start( ) throws BundleException

□ Starts this bundle. If the Framework implements the optional Start Level service and the current start level is less than this bundle's start level, then the Framework must persistently mark this bundle as started and delay the starting of this bundle until the Framework's current start level becomes equal or more than the bundle's start level.

Otherwise, the following steps are required to start a bundle:

1 If this bundle's state is `UNINSTALLED` then an `IllegalStateException` is thrown.
2 If this bundle's state is `STARTING` or `STOPPING` then this method will wait for this bundle to change state before continuing. If this does not occur in a reasonable time, a `BundleException` is thrown to indicate this bundle was unable to be started.
3 If this bundle's state is `ACTIVE` then this method returns immediately.
4 If this bundle's state is not `RESOLVED`, an attempt is made to resolve this bundle's package dependencies. If the Framework cannot resolve this bundle, a `BundleException` is thrown.
5 This bundle's state is set to `STARTING`.
6 The `BundleActivator.start`[p.97] method of this bundle's `BundleActivator`, if one is specified, is called. If the `BundleActivator` is invalid or throws an exception, this bundle's state is set back to `RESOLVED`.
Any services registered by the bundle will be unregistered.
Any services used by the bundle will be released.
Any listeners registered by the bundle will be removed.
A `BundleException` is then thrown.
7 If this bundle's state is `UNINSTALLED`, because the bundle was uninstalled while the `BundleActivator.start` method was running, a `BundleException` is thrown.
8 Since it is recorded that this bundle has been started, when the Framework is restarted this bundle will be automatically started.
9 This bundle's state is set to `ACTIVE`.
10 A bundle event of type `BundleEvent.STARTED`[p.108] is broadcast.

Preconditions

- `getState()` in {`INSTALLED`}, {`RESOLVED`}.

Postconditions, no exceptions thrown

- `getState()` in {`ACTIVE`}.
- `BundleActivator.start()` has been called and did not throw an exception.

Postconditions, when an exception is thrown

- `getState()` not in {`STARTING`}, {`ACTIVE`}.

Throws `BundleException` – If this bundle couldn't be started. This could be because a code dependency could not be resolved or the specified `BundleActivator` could not be loaded or threw an exception.

IllegalStateException – If this bundle has been uninstalled or this bundle tries to change its own state.

SecurityException – If the caller does not have the appropriate AdminPermisson, and the Java Runtime Environment supports permissions.

4.23.3.16 public void stop( ) throws BundleException

□ Stops this bundle.

The following steps are required to stop a bundle:

1 If this bundle's state is UNINSTALLED then an IllegalStateException is thrown.
2 If this bundle's state is STARTING or STOPPING then this method will wait for this bundle to change state before continuing. If this does not occur in a reasonable time, a BundleException is thrown to indicate this bundle was unable to be stopped.
3 If this bundle's state is not ACTIVE then this method returns immediately.
4 This bundle's state is set to STOPPING.
5 Since it is recorded that this bundle has been stopped, Framework is restarted this bundle will not be automatically started.
6 The BundleActivator.stop[p.98] method of this bundle's BundleActivator, if one is specified, is called. If this method throws an exception, it will continue to stop this bundle. A BundleException will be thrown after completion of the remaining steps.
7 Any services registered by this bundle must be unregistered.
8 Any services used by this bundle must be released.
9 Any listeners registered by this bundle must be removed.
10 If this bundle's state is UNINSTALLED, because the bundle was uninstalled while the BundleActivator.stop method was running, a BundleException must be thrown.
11 This bundle's state is set to RESOLVED.
12 A bundle event of type BundleEvent.STOPPED[p.108] is broadcast.

Preconditions

- getState() in {ACTIVE}.

Postconditions, no exceptions thrown

- getState() not in {ACTIVE, STOPPING}.
- BundleActivator.stop has been called and did not throw an exception.

Postconditions, when an exception is thrown

- None.

Throws BundleException – If this bundle's BundleActivator could not be loaded or threw an exception.

IllegalStateException – If this bundle has been uninstalled or this bundle tries to change its own state.

SecurityException – If the caller does not have the appropriate AdminPermission, and the Java Runtime Environment supports permissions.

4.23.3.17 public void uninstall( ) throws BundleException

□ Uninstalls this bundle.

This method causes the Framework to notify other bundles that this bundle is being uninstalled, and then puts this bundle into the UNINSTALLED state. The Framework will remove any resources related to this bundle that it is able to remove.

If this bundle has exported any packages, the Framework will continue to make these packages available to their importing bundles until the PackageAdmin.refreshPackages method has been called or the Framework is relaunched.

The following steps are required to uninstall a bundle:

1 If this bundle's state is UNINSTALLED then an IllegalStateException is thrown.
2 If this bundle's state is ACTIVE, STARTING or STOPPING, this bundle is stopped as described in the Bundle.stop method. If Bundle.stop throws an exception, a Framework event of type FrameworkEvent.ERROR[p.117] is broadcast containing the exception.
3 This bundle's state is set to UNINSTALLED.
4 A bundle event of type BundleEvent.UNINSTALLED[p.108] is broadcast.
5 This bundle and any persistent storage area provided for this bundle by the Framework are removed.

Preconditions

- getState() not in {UNINSTALLED}.

Postconditions, no exceptions thrown

- getState() in {UNINSTALLED}.
- This bundle has been uninstalled.

Postconditions, when an exception is thrown

- getState() not in {UNINSTALLED}.
- This Bundle has not been uninstalled.

Throws BundleException – If the uninstall failed. This can occur if another thread is attempting to change the bundle's state and does not complete in a timely manner.

IllegalStateException – If this bundle has been uninstalled or this bundle tries to change its own state.

SecurityException – If the caller does not have the appropriate AdminPermission, and the Java Runtime Environment supports permissions.

See Also stop()[p.94]

4.23.3.18 public void update( ) throws BundleException

□ Updates this bundle.

If this bundle's state is ACTIVE, it will be stopped before the update and started after the update successfully completes.

If the bundle being updated has exported any packages, these packages will not be updated. Instead, the previous package version will remain exported until the PackageAdmin.refreshPackages method has been has been called or the Framework is relaunched.

The following steps are required to update a bundle:

1 If this bundle's state is UNINSTALLED then an IllegalStateException is thrown.
2 If this bundle's state is ACTIVE, STARTING or STOPPING, the bundle is stopped as described in the Bundle.stop method. If Bundle.stop throws an exception, the exception is rethrown terminating the update.
3 The download location of the new version of this bundle is determined from either the bundle's Constants.BUNDLE_UPDATELOCATION[p.112] Manifest header (if available) or the bundle's original location.
4 The location is interpreted in an implementation dependent manner, typically as a URL, and the new version of this bundle is obtained from this location.
5 The new version of this bundle is installed. If the Framework is unable to install the new version of this bundle, the original version of this bundle will be restored and a BundleException will be thrown after completion of the remaining steps.
6 If the bundle has declared an Bundle-RequiredExecutionEnvironment header, then the listed execution environments must be verified against the installed execution environments. If they do not all match, the original version of this bundle will be restored and a BundleException will be thrown after completion of the remaining steps.
7 This bundle's state is set to INSTALLED.
8 If this bundle has not declared an Import-Package header in its Manifest file (specifically, this bundle does not depend on any packages from other bundles), this bundle's state may be set to RESOLVED.
9 If the new version of this bundle was successfully installed, a bundle event of type BundleEvent.UPDATED[p.108] is broadcast.
10 If this bundle's state was originally ACTIVE, the updated bundle is started as described in the Bundle.start method. If Bundle.start throws an exception, a Framework event of type FrameworkEvent.ERROR[p.117] is broadcast containing the exception.

Preconditions

- getState() not in {UNINSTALLED}.

Postconditions, no exceptions thrown

- getState() in {INSTALLED, RESOLVED, ACTIVE}.
- This bundle has been updated.

Postconditions, when an exception is thrown

- getState() in {INSTALLED, RESOLVED, ACTIVE}.
- Original bundle is still used; no update occurred.

Throws BundleException – If the update fails.

IllegalStateException – If this bundle has been uninstalled or this bundle tries to change its own state.

SecurityException – If the caller does not have the appropriate AdminPermission, and the Java Runtime Environment supports permissions.

See Also stop() [p.94], start() [p.93]

4.23.3.19 public void update(InputStream in) throws BundleException

in The InputStream from which to read the new bundle.

□ Updates this bundle from an InputStream.

This method performs all the steps listed in Bundle.update(), except the bundle will be read from the supplied InputStream, rather than a URL.

This method will always close the InputStream when it is done, even if an exception is thrown.

Throws BundleException – If the provided stream cannot be read or the update fails.

IllegalStateException – If this bundle has been uninstalled or this bundle tries to change its own state.

SecurityException – If the caller does not have the appropriate AdminPermission, and the Java Runtime Environment supports permissions.

See Also update() [p.95]

4.23.4 public interface BundleActivator

Customizes the starting and stopping of this bundle.

BundleActivator is an interface that may be implemented when this bundle is started or stopped. The Framework can create instances of this bundle's BundleActivator as required. If an instance's BundleActivator.start method executes successfully, it is guaranteed that the same instance's BundleActivator.stop method will be called when this bundle is to be stopped.

BundleActivator is specified through the Bundle-Activator Manifest header. A bundle can only specify a single BundleActivator in the Manifest file. The form of the Manifest header is:

Bundle-Activator: *class-name*

where class-name is a fully qualified Java classname.

The specified BundleActivator class must have a public constructor that takes no parameters so that a BundleActivator object can be created by Class.newInstance().

4.23.4.1 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.

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`[p.93]

4.23.4.2 public void stop(BundleContext context) throws Exception

context The execution context of the bundle being stopped.

□ Called when this bundle is stopped so the Framework can perform the bundle-specific activities necessary to stop the bundle. In general, this method should undo the work that the `BundleActivator.start` method started. There should be no active threads that were started by this bundle when this bundle returns. A stopped bundle should be stopped and should not call any Framework objects.

This method must complete and return to its caller in a timely manner.

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`[p.94]

4.23.5 public interface BundleContext

A bundle's execution context within the Framework. The context is used to grant access to other methods so that this bundle can interact with the Framework.

`BundleContext` methods allow a bundle to:

- Subscribe to events published by the Framework.
- Register service objects with the Framework service registry.
- Retrieve `ServiceReferences` from the Framework service registry.
- Get and release service objects for a referenced service.
- Install new bundles in the Framework.
- Get the list of bundles installed in the Framework.
- Get the `Bundle`[p.88] object for a bundle.
- Create `File` objects for files in a persistent storage area provided for the bundle by the Framework.

A `BundleContext` object will be created and provided to this bundle when it is started using the `BundleActivator.start`[p.97] method. The same `BundleContext` object will be passed to this bundle when it is stopped using the `BundleActivator.stop`[p.98] method. `BundleContext` is generally for the private use of this bundle and is not meant to be shared with other bundles in the OSGi environment. `BundleContext` is used when resolving `ServiceListeners` and `EventListener` objects.

The BundleContext object is only valid during an execution instance of this bundle; that is, during the period from when this bundle is called by BundleActivator.start until after this bundle is called and returns from BundleActivator.stop (or if BundleActivator.start terminates with an exception). If the BundleContext object is used subsequently, an IllegalStateException must be thrown. When this bundle is restarted, a new BundleContext object must be created.

The Framework is the only entity that can create BundleContext objects and they are only valid within the Framework that created them.

4.23.5.1 public void addBundleListener(BundleListener listener)

listener The BundleListener to be added.

- Adds the specified BundleListener object to this context bundle's list of listeners if not already present. See getBundle[p.100] for a definition of context bundle. BundleListener objects are notified when a bundle has a lifecycle state change.

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

Throws IllegalStateException – If this context bundle has stopped.

See Also BundleEvent[p.108], BundleListener[p.109]

4.23.5.2 public void addFrameworkListener(FrameworkListener listener)

listener The FrameworkListener object to be added.

- Adds the specified FrameworkListener object to this context bundle's list of listeners if not already present. See getBundle[p.100] for a definition of context bundle. FrameworkListeners are notified of general Framework events.

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

Throws IllegalStateException – If this context bundle has stopped.

See Also FrameworkEvent[p.117], FrameworkListener[p.119]

4.23.5.3 public void addServiceListener(ServiceListener listener, String filter) throws InvalidSyntaxException

listener The ServiceListener object to be added.

filter The filter criteria.

- Adds the specified ServiceListener object with the specified filter to this context bundle's list of listeners.

  See getBundle[p.100] for a definition of context bundle, and Filter[p.116] for a description of the filter syntax. ServiceListener objects are notified when a service has a lifecycle state change.

  If this context bundle's list of listeners already contains a listener l such that (l==listener), this method replaces that listener's filter (which may be null) with the specified one (which may be null).

The listener is called if the filter criteria is met. To filter based upon the class of the service, the filter should reference the Constants.OBJECTCLASS[p.115] property. If filter is null, all services are considered to match the filter.

When using a filter, it is possible that the ServiceEvents for the complete life cycle of a service will not be delivered to the listener. For example, if the filter only matches when the property x has the value 1, the listener will not be called if the service is registered with the property x not set to the value 1. Subsequently, when the service is modified setting property x to the value 1, the filter will match and the listener will be called with a ServiceEvent of type MODIFIED. Thus, the listener will not be called with a ServiceEvent of type REGISTERED.

If the Java Runtime Environment supports permissions, the ServiceListener object will be notified of a service event only if the bundle that is registering it has the ServicePermission to get the service using at least one of the named classes the service was registered under.

Throws InvalidSyntaxException – If filter contains an invalid filter string which cannot be parsed.

IllegalStateException – If this context bundle has stopped.

See Also ServiceEvent[p.121], ServiceListener[p.124], ServicePermission[p.124]

4.23.5.4 public void addServiceListener(ServiceListener listener)

listener The ServiceListener object to be added.

□ Adds the specified ServiceListener object to this context bundle's list of listeners.

This method is the same as calling BundleContext.addServiceListener(ServiceListener listener, String filter) with filter set to null.

Throws IllegalStateException – If this context bundle has stopped.

See Also addServiceListener(ServiceListener, String)[p.99]

4.23.5.5 public Filter createFilter(String filter) throws InvalidSyntaxException

filter The filter string.

□ Creates a Filter object. This Filter object may be used to match a ServiceReference object or a Dictionary object. See Filter[p.116] for a description of the filter string syntax.

If the filter cannot be parsed, an InvalidSyntaxException[p.119] will be thrown with a human readable message where the filter became unparsable.

Returns A Filter object encapsulating the filter string.

Throws InvalidSyntaxException – If filter contains an invalid filter string that cannot be parsed.

NullPointerException – If filter is null.

Since 1.1

4.23.5.6 public Bundle getBundle( )

- Returns the Bundle object for this context bundle.

  The context bundle is defined as the bundle that was assigned this BundleContext in its BundleActivator.

Returns The context bundle's Bundle object.

Throws IllegalStateException – If this context bundle has stopped.

4.23.5.7 public Bundle getBundle(long id)

id The identifier of the bundle to retrieve.

- Returns the bundle with the specified identifier.

Returns A Bundle object, or null if the identifier does not match any installed bundle.

4.23.5.8 public Bundle[] getBundles( )

- Returns a list of all installed bundles.

  This method returns a list of all bundles installed in the OSGi environment at the time of the call to this method. However, as the Framework is a very dynamic environment, bundles can be installed or uninstalled at anytime.

Returns An array of Bundle objects; one object per installed bundle.

4.23.5.9 public File getDataFile(String filename)

filename A relative name to the file to be accessed.

- Creates a File object for a file in the persistent storage area provided for the bundle by the Framework. This method will return null if the platform does not have file system support.

  A File object for the base directory of the persistent storage area provided for the context bundle by the Framework can be obtained by calling this method with an empty string ("") as filename. See getBundle[p.100] for a definition of context bundle.

  If the Java Runtime Environment supports permissions, the Framework will ensure that the bundle has the java.io.FilePermission with actions read, write, delete for all files (recursively) in the persistent storage area provided for the context bundle.

Returns A File object that represents the requested file or null if the platform does not have file system support.

Throws IllegalStateException – If the context bundle has stopped.

4.23.5.10 public String getProperty(String key)

key The name of the requested property.

- Returns the value of the specified property. If the key is not found in the Framework properties, the system properties are then searched. The method returns null if the property is not found.

  The Framework defines the following standard property keys:

  - Constants.FRAMEWORK_VERSION[p.114] - The OSGi Framework version.

- Constants.FRAMEWORK_VENDOR[p.114] - The Framework implementation vendor.
- Constants.FRAMEWORK_LANGUAGE[p.114] - The language being used. See ISO 639 for possible values.
- Constants.FRAMEWORK_OS_NAME[p.114] - The host computer operating system.
- Constants.FRAMEWORK_OS_VERSION[p.114] - The host computer operating system version number.
- Constants.FRAMEWORK_PROCESSOR[p.114] - The host computer processor name.

All bundles must have permission to read these properties.

Note: The last four standard properties are used by the Constants.BUNDLE_NATIVECODE[p.111] Manifest header's matching algorithm for selecting native language code.

Returns The value of the requested property, or null if the property is undefined.

Throws SecurityException – If the caller does not have the appropriate PropertyPermission to read the property, and the Java Runtime Environment supports permissions.

4.23.5.11 public Object getService(ServiceReference reference)

reference A reference to the service.

□ Returns the specified service object for a service.

A bundle's use of a service is tracked by the bundle's use count of that service. Each time a service's service object is returned by getService[p.102] the context bundle's use count for that service is incremented by one. Each time the service is released by ungetService[p.107] the context bundle's use count for that service is decremented by one.

When a bundle's use count for a service drops to zero, the bundle should no longer use that service. See getBundle[p.100] for a definition of context bundle.

This method will always return null when the service associated with this reference has been unregistered.

The following steps are required to get the service object:

1 If the service has been unregistered, null is returned.
2 The context bundle's use count for this service is incremented by one.
3 If the context bundle's use count for the service is currently one and the service was registered with an object implementing the ServiceFactory interface, the ServiceFactory.getService[p.123] method is called to create a service object for the context bundle. This service object is cached by the Framework. While the context bundle's use count for the service is greater than zero, subsequent calls to get the services's service object for the context bundle will return the cached service object.
If the service object returned by the ServiceFactory object is not an instanceof all the classes named when the service was registered or the ServiceFactory object throws an exception, null is returned and a Framework event of type FrameworkEvent.ERROR[p.117] is broadcast.
4 The service object for the service is returned.

Returns A service object for the service associated with `reference`, or `null` if the service is not registered or does not implement the classes under which it was registered in the case of a Service Factory.

Throws `SecurityException` – If the caller does not have the `ServicePermission` to get the service using at least one of the named classes the service was registered under, and the Java Runtime Environment supports permissions.

`IllegalStateException` – If the context bundle has stopped.

See Also `ungetService`[p.107] , `ServiceFactory`[p.123]

4.23.5.12 public ServiceReference getServiceReference(String clazz)

clazz The class name with which the service was registered.

□ Returns a `ServiceReference` object for a service that implements, and was registered under, the specified class.

This `ServiceReference` object is valid at the time of the call to this method, however as the Framework is a very dynamic environment, services can be modified or unregistered at anytime.

This method is the same as calling `getServiceReferences`[p.103] with a `null` filter string. It is provided as a convenience for when the caller is interested in any service that implements the specified class.

If multiple such services exist, the service with the highest ranking (as specified in its `Constants.SERVICE_RANKING`[p.116] property) is returned.

If there is a tie in ranking, the service with the lowest service ID (as specified in its `Constants.SERVICE_ID`[p.115] property); that is, the service that was registered first is returned.

Returns A `ServiceReference` object, or `null` if no services are registered which implement the named class.

See Also `getServiceReferences`[p.103]

4.23.5.13 public ServiceReference[] getServiceReferences(String clazz, String filter) throws InvalidSyntaxException

clazz The class name with which the service was registered, or `null` for all services.

filter The filter criteria.

□ Returns a list of `ServiceReference` objects. This method returns a list of `ServiceReference` objects for services which implement and were registered under the specified class and match the specified filter criteria.

The list is valid at the time of the call to this method, however as the Framework is a very dynamic environment, services can be modified or unregistered at anytime.

`filter` is used to select the registered service whose properties objects contain keys and values which satisfy the filter. See `Filter`[p.116] for a description of the filter string syntax.

If `filter` is `null`, all registered services are considered to match the filter.

If `filter` cannot be parsed, an `InvalidSyntaxException`[p.119] will be thrown with a human readable message where the filter became unparsable.

The following steps are required to select a service:

1 If the Java Runtime Environment supports permissions, the caller is checked for the ServicePermission to get the service with the specified class. If the caller does not have the correct permission, null is returned.
2 If the filter string is not null, the filter string is parsed and the set of registered services which satisfy the filter is produced. If the filter string is null, then all registered services are considered to satisfy the filter.
3 If clazz is not null, the set is further reduced to those services which are an instanceof and were registered under the specified class. The complete list of classes of which a service is an instance and which were specified when the service was registered is available from the service's Constants.OBJECTCLASS[p.115] property.
4 An array of ServiceReference to the selected services is returned.

Returns An array of ServiceReference objects, or null if no services are registered which satisfy the search.

Throws InvalidSyntaxException – If filter contains an invalid filter string which cannot be parsed.

4.23.5.14 public Bundle installBundle(String location) throws BundleException

location The location identifier of the bundle to install.

□ Installs the bundle from the specified location string. A bundle is obtained from location as interpreted by the Framework in an implementation dependent manner.

Every installed bundle is uniquely identified by its location string, typically in the form of a URL.

The following steps are required to install a bundle:

1 If a bundle containing the same location string is already installed, the Bundle object for that bundle is returned.
2 The bundle's content is read from the location string. If this fails, a BundleException[p.43] is thrown.
3 The bundle's Bundle-NativeCode dependencies are resolved. If this fails, a BundleException is thrown.
4 The bundle's associated resources are allocated. The associated resources minimally consist of a unique identifier, and a persistent storage area if the platform has file system support. If this step fails, a BundleException is thrown.
5 If the bundle has declared an Bundle-RequiredExecutionEnvironment header, then the listed execution environments must be verified against the installed execution environments. If they are not all present, a BundleException must be thrown.
6 The bundle's state is set to INSTALLED.
7 If the bundle has not declared an Import-Package Manifest header (that is, the bundle does not depend on any packages from other OSGi bundles), the bundle's state may be set to RESOLVED.
8 A bundle event of type BundleEvent.INSTALLED[p.108] is broadcast.
9 The Bundle object for the newly installed bundle is returned.

Postconditions, no exceptions thrown

- getState() in {INSTALLED, RESOLVED}.

- Bundle has a unique ID.

Postconditions, when an exception is thrown

- Bundle is not installed and no trace of the bundle exists.

Returns The `Bundle` object of the installed bundle.

Throws `BundleException` – If the installation failed.

`SecurityException` – If the caller does not have the appropriate `AdminPermission`, and the Java Runtime Environment supports permissions.

4.23.5.15 public Bundle installBundle(String location, InputStream in) throws BundleException

location The location identifier of the bundle to install.

in The `InputStream` object from which this bundle will be read.

□ Installs the bundle from the specified `InputStream` object.

This method performs all of the steps listed in `BundleContext.installBundle(String location)`, except that the bundle's content will be read from the `InputStream` object. The location identifier string specified will be used as the identity of the bundle.

This method must always close the `InputStream` object, even if an exception is thrown.

Returns The `Bundle` object of the installed bundle.

Throws `BundleException` – If the provided stream cannot be read or the installation failed.

`SecurityException` – If the caller does not have the appropriate `AdminPermission`, and the Java Runtime Environment supports permissions.

See Also `installBundle(java.lang.String)` [p.104]

4.23.5.16 public ServiceRegistration registerService(String[] clazzes, Object service, Dictionary properties)

clazzes The class names under which the service can be located. The class names in this array will be stored in the service's properties under the key `Constants.OBJECTCLASS` [p.115].

service The service object or a `ServiceFactory` object.

properties The properties for this service. The keys in the properties object must all be `String` objects. See `Constants` [p.110] for a list of standard service property keys. Changes should not be made to this object after calling this method. To update the service's properties the `ServiceRegistration.setProperties` [p.127] method must be called. `properties` may be `null` if the service has no properties.

□ Registers the specified service object with the specified properties under the specified class names into the Framework. A `ServiceRegistration` object is returned. The `ServiceRegistration` object is for the private use of the bundle registering the service and should not be shared with other bundles.

The registering bundle is defined to be the context bundle. See getBundle[p.100] for a definition of context bundle. Other bundles can locate the service by using either the getServiceReferences[p.103] or getServiceReference[p.103] method.

A bundle can register a service object that implements the ServiceFactory[p.123] interface to have more flexibility in providing service objects to other bundles.

The following steps are required to register a service:

1 If service is not a ServiceFactory, an IllegalArgumentException is thrown if service is not an instanceof all the classes named.
2 The Framework adds these service properties to the specified Dictionary (which may be null): a property named Constants.SERVICE_ID[p.115] identifying the registration number of the service, and a property named Constants.OBJECTCLASS[p.115] containing all the specified classes. If any of these properties have already been specified by the registering bundle, their values will be overwritten by the Framework.
3 The service is added to the Framework service registry and may now be used by other bundles.
4 A service event of type ServiceEvent.REGISTERED[p.122] is synchronously sent.
5 A ServiceRegistration object for this registration is returned.

Returns A ServiceRegistration object for use by the bundle registering the service to update the service's properties or to unregister the service.

Throws IllegalArgumentException – If one of the following is true: service is null. service is not a ServiceFactory object and is not an instance of all the named classes in clazzes. properties contains case variants of the same key name.

SecurityException – If the caller does not have the ServicePermission to register the service for all the named classes and the Java Runtime Environment supports permissions.

IllegalStateException – If this context bundle was stopped.

See Also ServiceRegistration[p.127], ServiceFactory[p.123]

4.23.5.17 public ServiceRegistration registerService(String clazz, Object service, Dictionary properties)

□ Registers the specified service object with the specified properties under the specified class name with the Framework.

This method is otherwise identical to registerService(java.lang.String[], java.lang.Object, java.util.Dictionary)[p.105] and is provided as a convenience when service will only be registered under a single class name. Note that even in this case the value of the service's Constants.OBJECTCLASS[p.115] property will be an array of strings, rather than just a single string.

See Also registerService(java.lang.String[], java.lang.Object, java.util.Dictionary)[p.105]

4.23.5.18 public void removeBundleListener(BundleListener listener)

listener The BundleListener object to be removed.

- Removes the specified BundleListener object from this context bundle's list of listeners. See getBundle[p.100] for a definition of context bundle.

  If listener is not contained in this context bundle's list of listeners, this method does nothing.

Throws IllegalStateException – If this context bundle has stopped.

4.23.5.19 public void removeFrameworkListener(FrameworkListener listener)

listener The FrameworkListener object to be removed.

- Removes the specified FrameworkListener object from this context bundle's list of listeners. See getBundle[p.100] for a definition of context bundle.

  If listener is not contained in this context bundle's list of listeners, this method does nothing.

Throws IllegalStateException – If this context bundle has stopped.

4.23.5.20 public void removeServiceListener(ServiceListener listener)

listener The ServiceListener to be removed.

- Removes the specified ServiceListener object from this context bundle's list of listeners. See getBundle[p.100] for a definition of context bundle.

  If listener is not contained in this context bundle's list of listeners, this method does nothing.

Throws IllegalStateException – If this context bundle has stopped.

4.23.5.21 public boolean ungetService(ServiceReference reference)

reference A reference to the service to be released.

- Releases the service object referenced by the specified ServiceReference object. If the context bundle's use count for the service is zero, this method returns false. Otherwise, the context bundle's use count for the service is decremented by one. See getBundle[p.100] for a definition of context bundle.

  The service's service object should no longer be used and all references to it should be destroyed when a bundle's use count for the service drops to zero.

  The following steps are required to unget the service object:

  1 If the context bundle's use count for the service is zero or the service has been unregistered, false is returned.
  2 The context bundle's use count for this service is decremented by one.
  3 If the context bundle's use count for the service is currently zero and the service was registered with a ServiceFactory object, the ServiceFactory.ungetService[p.123] method is called to release the service object for the context bundle.
  4 true is returned.

Returns false if the context bundle's use count for the service is zero or if the service has been unregistered; true otherwise.

Throws IllegalStateException – If the context bundle has stopped.

See Also getService[p.102], ServiceFactory[p.123]

4.23.6 public class BundleEvent extends EventObject

A Framework event describing a bundle lifecycle change.

BundleEvent objects are delivered to BundleListener objects when a change occurs in a bundle's lifecycle. A type code is used to identify the event type for future extendability.

OSGi reserves the right to extend the set of types.

4.23.6.1 public static final int INSTALLED = 1

This bundle has been installed.

The value of INSTALLED is 0x00000001.

See Also BundleContext.installBundle[p.104]

4.23.6.2 public static final int STARTED = 2

This bundle has been started.

The value of STARTED is 0x00000002.

See Also Bundle.start[p.93]

4.23.6.3 public static final int STOPPED = 4

This bundle has been stopped.

The value of STOPPED is 0x00000004.

See Also Bundle.stop[p.94]

4.23.6.4 public static final int UNINSTALLED = 16

This bundle has been uninstalled.

The value of UNINSTALLED is 0x00000010.

See Also Bundle.uninstall[p.94]

4.23.6.5 public static final int UPDATED = 8

This bundle has been updated.

The value of UPDATED is 0x00000008.

See Also Bundle.update[p.95]

4.23.6.6 public BundleEvent(int type, Bundle bundle)

type The event type.

bundle The bundle which had a lifecycle change.

- Creates a bundle event of the specified type.

4.23.6.7 public Bundle getBundle( )

- Returns the bundle which had a lifecycle change. This bundle is the source of the event.

Returns A bundle that had a change occur in its lifecycle.

4.23.6.8 public int getType( )

□ Returns the type of lifecyle event. The type values are:

- INSTALLED[p.108]
- STARTED[p.108]
- STOPPED[p.108]
- UPDATED[p.108]
- UNINSTALLED[p.108]

Returns The type of lifecycle event.

4.23.7 public class BundleException extends Exception

A Framework exception used to indicate that a bundle lifecycle problem occurred.

`BundleException` object is created by the Framework to denote an exception condition in the lifecycle of a bundle. `BundleException`s should not be created by bundle developers.

4.23.7.1 public BundleException(String msg, Throwable throwable)

msg The associated message.

throwable The nested exception.

□ Creates a `BundleException` that wraps another exception.

4.23.7.2 public BundleException(String msg)

msg The message.

□ Creates a `BundleException` object with the specified message.

4.23.7.3 public Throwable getNestedException( )

□ Returns any nested exceptions included in this exception.

Returns The nested exception; `null` if there is no nested exception.

4.23.8 public interface BundleListener extends EventListener

A `BundleEvent` listener.

`BundleListener` is a listener interface that may be implemented by a bundle developer.

A `BundleListener` object is registered with the Framework using the `BundleContext.addBundleListener`[p.99] method. `BundleListener`s are called with a `BundleEvent` object when a bundle has been installed, started, stopped, updated, or uninstalled.

See Also `BundleEvent`[p.108]

4.23.8.1 public void bundleChanged(BundleEvent event)

event The `BundleEvent`.

□ Receives notification that a bundle has had a lifecycle change.

4.23.9 public interface Configurable

Supports a configuration object.

`Configurable` is an interface that should be used by a bundle developer in support of a configurable service. Bundles that need to configure a service may test to determine if the service object is an `instanceof Configurable`.

4.23.9.1 public Object getConfigurationObject( )

□ Returns this service's configuration object.

Services implementing `Configurable` should take care when returning a service configuration object since this object is probably sensitive.

If the Java Runtime Environment supports permissions, it is recommended that the caller is checked for the appropriate permission before returning the configuration object. It is recommended that callers possessing the appropriate `AdminPermission`[p.87] always be allowed to get the configuration object.

Returns The configuration object for this service.

Throws `SecurityException` – If the caller does not have an appropriate permission and the Java Runtime Environment supports permissions.

4.23.10 public interface Constants

Defines standard names for the OSGi environment property, service property, and Manifest header attribute keys.

The values associated with these keys are of type `java.lang.String`, unless otherwise indicated.

See Also `Bundle.getHeaders`[p.90], `BundleContext.getProperty`[p.101], `BundleContext.registerService`[p.105]

Since 1.1

4.23.10.1 public static final String BUNDLE_ACTIVATOR = "Bundle-Activator"

Manifest header attribute (named "Bundle-Activator") identifying the bundle's activator class.

If present, this header specifies the name of the bundle resource class that implements the `BundleActivator` interface and whose `start` and `stop` methods are called by the Framework when the bundle is started and stopped, respectively.

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

4.23.10.2 public static final String BUNDLE_CATEGORY = "Bundle-Category"

Manifest header (named "Bundle-Category") identifying the bundle's category.

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

4.23.10.3 public static final String BUNDLE_CLASSPATH = "Bundle-ClassPath"

Manifest header (named "Bundle-ClassPath") identifying a list of nested JAR files, which are bundle resources used to extend the bundle's classpath.

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

4.23.10.4 public static final String BUNDLE_CONTACTADDRESS = "Bundle-ContactAddress"

Manifest header (named "Bundle-ContactAddress") identifying the contact address where problems with the bundle may be reported; for example, an email address.

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

4.23.10.5 public static final String BUNDLE_COPYRIGHT = "Bundle-Copyright"

Manifest header (named "Bundle-Copyright") identifying the bundle's copyright information, which may be retrieved from the `Dictionary` object returned by the `Bundle.getHeaders` method.

4.23.10.6 public static final String BUNDLE_DESCRIPTION = "Bundle-Description"

Manifest header (named "Bundle-Description") containing a brief description of the bundle's functionality.

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

4.23.10.7 public static final String BUNDLE_DOCURL = "Bundle-DocURL"

Manifest header (named "Bundle-DocURL") identifying the bundle's documentation URL, from which further information about the bundle may be obtained.

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

4.23.10.8 public static final String BUNDLE_NAME = "Bundle-Name"

Manifest header (named "Bundle-Name") identifying the bundle's name.

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

4.23.10.9 public static final String BUNDLE_NATIVECODE = "Bundle-NativeCode"

Manifest header (named "Bundle-NativeCode") identifying a number of hardware environments and the native language code libraries that the bundle is carrying for each of these environments.

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

4.23.10.10 public static final String BUNDLE_NATIVECODE_LANGUAGE = "language"

Manifest header attribute (named "language") identifying the language in which the native bundle code is written specified in the Bundle-NativeCode Manifest header. See ISO 639 for possible values.

The attribute value is encoded in the Bundle-NativeCode Manifest header like:

```
Bundle-NativeCode: http.so ; language=nl_be ...
```

4.23.10.11 public static final String BUNDLE_NATIVECODE_OSNAME = "osname"

Manifest header attribute (named "osname") identifying the operating system required to run native bundle code specified in the Bundle-NativeCode Manifest header).

The attribute value is encoded in the Bundle-NativeCode Manifest header like:

```
Bundle-NativeCode: http.so ; osname=Linux ...
```

4.23.10.12 public static final String BUNDLE_NATIVECODE_OSVERSION = "osversion"

Manifest header attribute (named "osversion") identifying the operating system version required to run native bundle code specified in the Bundle-NativeCode Manifest header).

The attribute value is encoded in the Bundle-NativeCode Manifest header like:

```
Bundle-NativeCode: http.so ; osversion="2.34" ...
```

4.23.10.13 public static final String BUNDLE_NATIVECODE_PROCESSOR = "processor"

Manifest header attribute (named "processor") identifying the processor required to run native bundle code specified in the Bundle-NativeCode Manifest header).

The attribute value is encoded in the Bundle-NativeCode Manifest header like:

```
Bundle-NativeCode: http.so ; processor=x86 ...
```

4.23.10.14 public static final String BUNDLE_REQUIREDEXECUTIONENVIRONMENT = "Bundle-RequiredExecutionEnvironment"

Manifest header (named "Bundle-RequiredExecutionEnvironment") identifying the required execution environment for the bundle. The service platform may run this bundle if any of the execution environments named in this header matches one of the execution environments it implements.

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

Since 1.2

4.23.10.15 public static final String BUNDLE_UPDATELOCATION = "Bundle-

UpdateLocation"

Manifest header (named "Bundle-UpdateLocation") identifying the location from which a new bundle version is obtained during a bundle update operation.

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

4.23.10.16 public static final String BUNDLE_VENDOR = "Bundle-Vendor"

Manifest header (named "Bundle-Vendor") identifying the bundle's vendor.

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

4.23.10.17 public static final String BUNDLE_VERSION = "Bundle-Version"

Manifest header (named "Bundle-Version") identifying the bundle's version.

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

4.23.10.18 public static final String DYNAMICIMPORT_PACKAGE = "DynamicImport-Package"

Manifest header (named "DynamicImport-Package") identifying the names of the packages that the bundle may dynamically import during execution.

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

Since 1.2

4.23.10.19 public static final String EXPORT_PACKAGE = "Export-Package"

Manifest header (named "Export-Package") identifying the names (and optionally version numbers) of the packages that the bundle offers to the Framework for export.

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

4.23.10.20 public static final String EXPORT_SERVICE = "Export-Service"

Manifest header (named "Export-Service") identifying the fully qualified class names of the services that the bundle may register (used for informational purposes only).

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

4.23.10.21 public static final String FRAMEWORK_EXECUTIONENVIRONMENT = "org.osgi.framework.executionenvironment"

Framework environment property (named "org.osgi.framework.executionenvironment") identifying execution environments provided by the Framework.

The value of this property may be retrieved by calling the `BundleContext.getProperty` method.

Since 1.2

4.23.10.22 public static final String FRAMEWORK_LANGUAGE = "org.osgi.framework.language"

Framework environment property (named "org.osgi.framework.language") identifying the Framework implementation language (see ISO 639 for possible values).

The value of this property may be retrieved by calling the `BundleContext.getProperty` method.

4.23.10.23 public static final String FRAMEWORK_OS_NAME = "org.osgi.framework.os.name"

Framework environment property (named "org.osgi.framework.os.name") identifying the Framework host-computer's operating system.

The value of this property may be retrieved by calling the `BundleContext.getProperty` method.

4.23.10.24 public static final String FRAMEWORK_OS_VERSION = "org.osgi.framework.os.version"

Framework environment property (named "org.osgi.framework.os.version") identifying the Framework host-computer's operating system version number.

The value of this property may be retrieved by calling the `BundleContext.getProperty` method.

4.23.10.25 public static final String FRAMEWORK_PROCESSOR = "org.osgi.framework.processor"

Framework environment property (named "org.osgi.framework.processor") identifying the Framework host-computer's processor name.

The value of this property may be retrieved by calling the `BundleContext.getProperty` method.

4.23.10.26 public static final String FRAMEWORK_VENDOR = "org.osgi.framework.vendor"

Framework environment property (named "org.osgi.framework.vendor") identifying the Framework implementation vendor.

The value of this property may be retrieved by calling the `BundleContext.getProperty` method.

4.23.10.27 public static final String FRAMEWORK_VERSION = "org.osgi.framework.version"

Framework environment property (named "org.osgi.framework.version") identifying the Framework version.

The value of this property may be retrieved by calling the `BundleContext.getProperty` method.

4.23.10.28 public static final String IMPORT_PACKAGE = "Import-Package"

Manifest header (named "Import-Package") identifying the names (and optionally, version numbers) of the packages that the bundle is dependent on.

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

4.23.10.29 public static final String IMPORT_SERVICE = "Import-Service"

Manifest header (named "Import-Service") identifying the fully qualified class names of the services that the bundle requires (used for informational purposes only).

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

4.23.10.30 public static final String OBJECTCLASS = "objectClass"

Service property (named "objectClass") identifying all of the class names under which a service was registered in the Framework (of type `java.lang.String[]`).

This property is set by the Framework when a service is registered.

4.23.10.31 public static final String PACKAGE_SPECIFICATION_VERSION = "specification-version"

Manifest header attribute (named "specification-version") identifying the version of a package specified in the Export-Package or Import-Package Manifest header.

The attribute value is encoded in the Export-Package or Import-Package Manifest header like:

```
Import-Package: org.osgi.framework ; specification-ver-
sion="1.1"
```

4.23.10.32 public static final String SERVICE_DESCRIPTION = "service.description"

Service property (named "service.description") identifying a service's description.

This property may be supplied in the properties `Dictionary` object passed to the `BundleContext.registerService` method.

4.23.10.33 public static final String SERVICE_ID = "service.id"

Service property (named "service.id") identifying a service's registration number (of type `java.lang.Long`).

The value of this property is assigned by the Framework when a service is registered. The Framework assigns a unique value that is larger than all previously assigned values since the Framework was started. These values are NOT persistent across restarts of the Framework.

4.23.10.34 public static final String SERVICE_PID = "service.pid"

Service property (named "service.pid") identifying a service's persistent identifier.

This property may be supplied in the propertiesDictionary object passed to the BundleContext.registerService method.

A service's persistent identifier uniquely identifies the service and persists across multiple Framework invocations.

By convention, every bundle has its own unique namespace, starting with the bundle's identifier (see Bundle.getBundleId[p.90]) and followed by a dot (.). A bundle may use this as the prefix of the persistent identifiers for the services it registers.

4.23.10.35 public static final String SERVICE_RANKING = "service.ranking"

Service property (named "service.ranking") identifying a service's ranking number (of type java.lang.Integer).

This property may be supplied in the properties Dictionary object passed to the BundleContext.registerService method.

The service ranking is used by the Framework to determine the *default* service to be returned from a call to the BundleContext.getServiceReference[p.103] method: If more than one service implements the specified class, the ServiceReference object with the highest ranking is returned.

The default ranking is zero (0). A service with a ranking of Integer.MAX_VALUE is very likely to be returned as the default service, whereas a service with a ranking of Integer.MIN_VALUE is very unlikely to be returned.

If the supplied property value is not of type java.lang.Integer, it is deemed to have a ranking value of zero.

4.23.10.36 public static final String SERVICE_VENDOR = "service.vendor"

Service property (named "service.vendor") identifying a service's vendor.

This property may be supplied in the properties Dictionary object passed to the BundleContext.registerService method.

4.23.10.37 public static final String SYSTEM_BUNDLE_LOCATION = "System Bundle"

Location identifier of the OSGi *system bundle*, which is defined to be "System Bundle".

4.23.11 public interface Filter

An RFC 1960-based Filter.

Filter objects can be created by calling BundleContext.createFilter[p.100] with the chosen filter string.

A Filter object can be used numerous times to determine if the match argument matches the filter string that was used to create the Filter object.

Some examples of LDAP filters are:

```
"(cn=Babs Jensen)"
"(!(cn=Tim Howes))"
"(&(" + Constants.OBJECTCLASS +
```

```
"=Person)(|(sn=Jensen)(cn=Babs J*)))"
   "(o=univ*of*mich*)"
```

Since 1.1

4.23.11.1 public boolean equals(Object obj)

obj The object to compare against this `Filter` object.

Compares this `Filter` object to another object.

Returns If the other object is a `Filter` object, then returns `this.toString().equals(obj.toString()`; `false` otherwise.

4.23.11.2 public int hashCode( )

Returns the hashCode for this `Filter` object.

Returns The hashCode of the filter string; that is, `this.toString().hashCode()`.

4.23.11.3 public boolean match(ServiceReference reference)

reference The reference to the service whose properties are used in the match.

Filter using a service's properties.

The filter is executed using properties of the referenced service.

Returns `true` if the service's properties match this filter; `false` otherwise.

4.23.11.4 public boolean match(Dictionary dictionary)

dictionary The `Dictionary` object whose keys are used in the match.

Filter using a `Dictionary` object. The Filter is executed using the `Dictionary` object's keys and values.

Returns `true` if the `Dictionary` object's keys and values match this filter; `false` otherwise.

Throws `IllegalArgumentException` – If `dictionary` contains case variants of the same key name.

4.23.11.5 public String toString( )

Returns this `Filter` object's filter string.

The filter string is normalized by removing whitespace which does not affect the meaning of the filter.

Returns Filter string.

4.23.12 public class FrameworkEvent extends EventObject

A general Framework event.

`FrameworkEvent` is the event class used when notifying listeners of general events occuring within the OSGI environment. A type code is used to identify the event type for future extendability.

OSGi reserves the right to extend the set of event types.

4.23.12.1 public static final int ERROR = 2

An error has occurred.

There was an error associated with a bundle.

The value of ERROR is 0x00000002.

4.23.12.2 public static final int PACKAGES_REFRESHED = 4

A PackageAdmin.refreshPackage operation has completed.

This event is broadcast when the Framework has completed the refresh packages operation initiated by a call to the PackageAdmin.refreshPackages method.

The value of PACKAGES_REFRESHED is 0x00000004.

See Also `org.osgi.service.packageadmin.PackageAdmin.refreshPackages`

Since 1.2

4.23.12.3 public static final int STARTED = 1

The Framework has started.

This event is broadcast when the Framework has started after all installed bundles that are marked to be started have been started and the Framework has reached the intitial start level.

The value of STARTED is 0x00000001.

See Also `org.osgi.service.startlevel.StartLevel`

4.23.12.4 public static final int STARTLEVEL_CHANGED = 8

A StartLevel.setStartLevel operation has completed.

This event is broadcast when the Framework has completed changing the active start level initiated by a call to the StartLevel.setStartLevel method.

The value of STARTLEVEL_CHANGED is 0x00000008.

See Also `org.osgi.service.startlevel.StartLevel`

Since 1.2

4.23.12.5 public FrameworkEvent(int type, Object source)

type The event type.

source The event source object. This may not be `null`.

- Creates a Framework event.

Deprecated Since 1.2. This constructor is deprecated in favor of using the other constructor with the System Bundle as the event source.

4.23.12.6 public FrameworkEvent(int type, Bundle bundle, Throwable throwable)

type The event type.

bundle The event source.

throwable The related exception. This argument may be `null` if there is no related exception.

- Creates a Framework event regarding the specified bundle.

4.23.12.7 public Bundle getBundle( )

- Returns the bundle associated with the event. This bundle is also the source of the event.

Returns The bundle associated with the event.

4.23.12.8 public Throwable getThrowable( )

- Returns the exception associated with the event.

  If the event type is ERROR, this method returns the exception related to the error.

Returns An exception if an event of type ERROR or null.

4.23.12.9 public int getType( )

- Returns the type of bundle state change.

  The type values are:

  - STARTED[p.118]
  - ERROR[p.117]
  - PACKAGES_REFRESHED[p.118]
  - STARTLEVEL_CHANGED[p.118]

Returns The type of state change.

4.23.13 public interface FrameworkListener extends EventListener

A FrameworkEvent listener.

FrameworkListener is a listener interface that may be implemented by a bundle developer. A FrameworkListener object is registered with the Framework using the BundleContext.addFrameworkListener[p.99] method. FrameworkListener objects are called with a FrameworkEvent objects when the Framework starts and when asynchronous errors occur.

See Also FrameworkEvent[p.117]

4.23.13.1 public void frameworkEvent(FrameworkEvent event)

event The FrameworkEvent object.

- Receives notification of a general FrameworkEvent object.

4.23.14 public class InvalidSyntaxException extends Exception

A Framework exception.

An InvalidSyntaxException object indicates that a filter string parameter has an invalid syntax and cannot be parsed.

See Filter[p.116] for a description of the filter string syntax.

4.23.14.1 public InvalidSyntaxException(String msg, String filter)

msg The message.

filter The invalid filter string.

- Creates an exception of type InvalidSyntaxException.

  This method creates an InvalidSyntaxException object with the specified message and the filter string which generated the exception.

4.23.14.2 public String getFilter( )

- Returns the filter string that generated the InvalidSyntaxException object.

Returns The invalid filter string.

See Also BundleContext.getServiceReferences[p.103], BundleContext.addServiceListener[p.99]

4.23.15 public final class PackagePermission extends BasicPermission

A bundle's authority to import or export a package.

A package is a dot-separated string that defines a fully qualified Java package.

For example:

```
org.osgi.service.http
```

PackagePermission has two actions: EXPORT and IMPORT. The EXPORT action implies the IMPORT action.

4.23.15.1 public static final String EXPORT = "export"

The action string export.

4.23.15.2 public static final String IMPORT = "import"

The action string import.

4.23.15.3 public PackagePermission(String name, String actions)

name Package name.

actions EXPORT, IMPORT (canonical order).

- Defines the authority to import and/or export a package within the OSGi environment.

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

  ```
  org.osgi.service.http
  javax.servlet.*
  *
  ```

  Package Permissions are granted over all possible versions of a package. A bundle that needs to export a package must have the appropriate PackagePermission for that package; similarly, a bundle that needs to import a package must have the appropriate PackagePermssion for that package.

  Permission is granted for both classes and resources.

4.23.15.4 public boolean equals(Object obj)

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

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

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

4.23.15.5 public String getActions( )

- Returns the canonical string representation of the PackagePermission actions.

  Always returns present PackagePermission actions in the following order: EXPORT, IMPORT.

Returns Canonical string representation of the PackagePermission actions.

4.23.15.6 public int hashCode( )

- Returns the hash code value for this object.

Returns A hash code value for this object.

4.23.15.7 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 package name of the target is implied by the package name of this object. The list of PackagePermission actions must either match or allow for the list of the target object to imply the target PackagePermission action.

  The permission to export a package implies the permission to import the named package.

```
x.y.*,"export" -> x.y.z,"export" is true
*,"import" -> x.y, "import"      is true
*,"export" -> x.y, "import"      is true
x.y,"export" -> x.y.z, "export"  is false
```

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

4.23.15.8 public PermissionCollection newPermissionCollection( )

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

Returns A new PermissionCollection object.

4.23.16 public class ServiceEvent extends EventObject

A service lifecycle change event.

ServiceEvent objects are delivered to a ServiceListener objects when a change occurs in this service's lifecycle. A type code is used to identify the event type for future extendability.

OSGi reserves the right to extend the set of types.

See Also ServiceListener[p.124]

4.23.16.1 public static final int MODIFIED = 2

The properties of a registered service have been modified.

This event is synchronously delivered after the service properties have been modified.

The value of MODIFIED is 0x00000002.

See Also ServiceRegistration.setProperties[p.127]

4.23.16.2 public static final int REGISTERED = 1

This service has been registered.

This event is synchronously delivered after the service has been registered with the Framework.

The value of REGISTERED is 0x00000001.

See Also BundleContext.registerService[p.105]

4.23.16.3 public static final int UNREGISTERING = 4

This service is in the process of being unregistered.

This event is synchronously delivered before the service has completed unregistering.

If a bundle is using a service that is UNREGISTERING, the bundle should release its use of the service when it receives this event. If the bundle does not release its use of the service when it receives this event, the Framework will automatically release the bundle's use of the service while completing the service unregistration operation.

The value of UNREGISTERING is 0x00000004.

See Also ServiceRegistration.unregister[p.127], BundleContext.ungetService[p.107]

4.23.16.4 public ServiceEvent(int type, ServiceReference reference)

type The event type.

reference A ServiceReference object to the service that had a lifecycle change.

☐ Creates a new service event object.

4.23.16.5 public ServiceReference getServiceReference( )

☐ Returns a reference to the service that had a change occur in its lifecycle.

This reference is the source of the event.

Returns Reference to the service that had a lifecycle change.

4.23.16.6 public int getType( )

☐ Returns the type of event. The event type values are:

- REGISTERED[p.122]
- MODIFIED[p.122]
- UNREGISTERING[p.122]

Returns Type of service lifecycle change.

4.23.17 public interface ServiceFactory

Allows services to provide customized service objects in the OSGi environment.

When registering a service, a `ServiceFactory` object can be used instead of a service object, so that the bundle developer can gain control of the specific service object granted to a bundle that is using the service.

When this happens, the `BundleContext.getService(ServiceReference)` method calls the `ServiceFactory.getService` method to create a service object specifically for the requesting bundle. The service object returned by the `ServiceFactory` object is cached by the Framework until the bundle releases its use of the service.

When the bundle's use count for the service equals zero (including the bundle stopping or the service being unregistered), the `ServiceFactory.ungetService` method is called.

`ServiceFactory` objects are only used by the Framework and are not made available to other bundles in the OSGi environment.

See Also `BundleContext.getService`[p.102]

4.23.17.1 public Object getService(Bundle bundle, ServiceRegistration registration)

bundle The bundle using the service.

registration The `ServiceRegistration` object for the service.

□ Creates a new service object.

The Framework invokes this method the first time the specified `bundle` requests a service object using the `BundleContext.getService(ServiceReference)` method. The service factory can then return a specific service object for each bundle.

The Framework caches the value returned (unless it is `null`), and will return the same service object on any future call to `BundleContext.getService` from the same bundle.

The Framework will check if the returned service object is an instance of all the classes named when the service was registered. If not, then `null` is returned to the bundle.

Returns A service object that must be an instance of all the classes named when the service was registered.

See Also `BundleContext.getService`[p.102]

4.23.17.2 public void ungetService(Bundle bundle, ServiceRegistration registration, Object service)

bundle The bundle releasing the service.

registration The `ServiceRegistration` object for the service.

service The service object returned by a previous call to the `ServiceFactory.getService` method.

- Releases a service object.

  The Framework invokes this method when a service has been released by a bundle. The service object may then be destroyed.

See Also BundleContext.ungetService[p.107]

4.23.18 public interface ServiceListener extends EventListener

A ServiceEvent listener.

ServiceListener is a listener interface that may be implemented by a bundle developer.

A ServiceListener object is registered with the Framework using the BundleContext.addServiceListener method. ServiceListener objects are called with a ServiceEvent object when a service has been registered or modified, or is in the process of unregistering.

ServiceEvent object delivery to ServiceListener objects is filtered by the filter specified when the listener was registered. If the Java Runtime Environment supports permissions, then additional filtering is done. ServiceEvent objects are only delivered to the listener if the bundle which defines the listener object's class has the appropriate ServicePermission to get the service using at least one of the named classes the service was registered under.

See Also ServiceEvent[p.121], ServicePermission[p.124]

4.23.18.1 public void serviceChanged(ServiceEvent event)

event The ServiceEvent object.

- Receives notification that a service has had a lifecycle change.

4.23.19 public final class ServicePermission extends BasicPermission

Indicates a bundle's authority to register or get a service.

- The ServicePermission.REGISTER action allows a bundle to register a service on the specified names.
- The ServicePermission.GET action allows a bundle to detect a service and get it.

ServicePermission to get the specific service.

4.23.19.1 public static final String GET = "get"

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

4.23.19.2 public static final String REGISTER = "register"

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

4.23.19.3 public ServicePermission(String name, String actions)

name class name

actions get, register (canonical order)

- Create a new ServicePermission.

  The name of the service is specified as a fully qualified class name.

  ```
  ClassName ::= <class name> | <class name ending in ".*">
  ```

  Examples:

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

  There are two possible actions: `get` and `register`. The `get` permission allows the owner of this permission to obtain a service with this name. The `register` permission allows the bundle to register a service under that name.

4.23.19.4 public boolean equals(Object obj)

obj The object to test for equality.

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

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

4.23.19.5 public String getActions( )

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

Returns The canonical string representation of the actions.

4.23.19.6 public int hashCode( )

- Returns the hash code value for this object.

Returns Hash code value for this object.

4.23.19.7 public boolean implies(Permission p)

p The target permission to check.

- Determines if a `ServicePermission` object "implies" the specified permission.

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

4.23.19.8 public PermissionCollection newPermissionCollection( )

- Returns a new `PermissionCollection` object for storing `ServicePermission objects`.

Returns A new `PermissionCollection` object suitable for storing `ServicePermission` objects.

4.23.20 public interface ServiceReference

A reference to a service.

The Framework returns `ServiceReference` objects from the `BundleContext.getServiceReference` and `BundleContext.getServiceReferences` methods.

A ServiceReference may be shared between bundles and can be used to examine the properties of the service and to get the service object.

Every service registered in the Framework has a unique ServiceRegistration object and may have multiple, distinct ServiceReference objects referring to it. ServiceReference objects associated with a ServiceRegistration object have the same hashCode and are considered equal (more specifically, their equals() method will return true when compared).

If the same service object is registered multiple times, ServiceReference objects associated with different ServiceRegistration objects are not equal.

See Also BundleContext.getServiceReference[p.103], BundleContext.getServiceReferences[p.103], BundleContext.getService[p.102]

4.23.20.1 public Bundle getBundle( )

- Returns the bundle that registered the service referenced by this ServiceReference object.

  This method will always return null when the service has been unregistered. This can be used to determine if the service has been unregistered.

Returns The bundle that registered the service referenced by this ServiceReference object; null if that service has already been unregistered.

See Also BundleContext.registerService[p.105]

4.23.20.2 public Object getProperty(String key)

key The property key.

- Returns the property value to which the specified property key is mapped in the properties Dictionary object of the service referenced by this ServiceReference object.

  Property keys are case-insensitive.

  This method must continue to return property values after the service has been unregistered. This is so references to unregistered services (for example, ServiceReference objects stored in the log) can still be interrogated.

Returns The property value to which the key is mapped; null if there is no property named after the key.

4.23.20.3 public String[] getPropertyKeys( )

- Returns an array of the keys in the properties Dictionary object of the service referenced by this ServiceReference object.

  This method will continue to return the keys after the service has been unregistered. This is so references to unregistered services (for example, ServiceReference objects stored in the log) can still be interrogated.

  This method is *case-preserving*; this means that every key in the returned array must have the same case as the corresponding key in the properties Dictionary that was passed to the BundleContext.registerService[p.105] or ServiceRegistration.setProperties[p.127] methods.

Returns An array of property keys.

4.23.20.4 public Bundle[] getUsingBundles( )

- Returns the bundles that are using the service referenced by this ServiceReference object. Specifically, this method returns the bundles whose usage count for that service is greater than zero.

Returns An array of bundles whose usage count for the service referenced by this ServiceReference object is greater than zero; null if no bundles are currently using that service.

Since 1.1

4.23.21 public interface ServiceRegistration

A registered service.

The Framework returns a ServiceRegistration object when a BundleContext.registerService method is successful. The ServiceRegistration object is for the private use of the registering bundle and should not be shared with other bundles.

The ServiceRegistration object may be used to update the properties of the service or to unregister the service.

See Also BundleContext.registerService[p.105]

4.23.21.1 public ServiceReference getReference( )

- Returns a ServiceReference object for a service being registered.

  The ServiceReference object may be shared with other bundles.

Returns ServiceReference object.

Throws IllegalStateException – If this ServiceRegistration object has already been unregistered.

4.23.21.2 public void setProperties(Dictionary properties)

properties The properties for this service. See Constants[p.110] for a list of standard service property keys. Changes should not be made to this object after calling this method. To update the service's properties this method should be called again.

- Updates the properties associated with a service.

  The Constants.OBJECTCLASS[p.115] and Constants.SERVICE_ID[p.115] keys cannot be modified by this method. These values are set by the Framework when the service is registered in the OSGi environment.

  The following steps are required to modify service properties:

  1 The service's properties are replaced with the provided properties.
  2 A service event of type ServiceEvent.MODIFIED[p.122] is synchronously sent.

Throws IllegalStateException – If this ServiceRegistration object has already been unregistered.

IllegalArgumentException – If properties contains case variants of the same key name.

4.23.21.3 public void unregister( )

□ Unregisters a service. Remove a ServiceRegistration object from the Framework service registry. All ServiceReference objects associated with this ServiceRegistration object can no longer be used to interact with the service.

The following steps are required to unregister a service:

1 The service is removed from the Framework service registry so that it can no longer be used. ServiceReference objects for the service may no longer be used to get a service object for the service.
2 A service event of type ServiceEvent.UNREGISTERING[p.122] is synchronously sent so that bundles using this service can release their use of it.
3 For each bundle whose use count for this service is greater than zero:
The bundle's use count for this service is set to zero.
If the service was registered with a ServiceFactory[p.123] object, the ServiceFactory.ungetService method is called to release the service object for the bundle.

Throws IllegalStateException – If this ServiceRegistration object has already been unregistered.

See Also BundleContext.ungetService[p.107], ServiceFactory.ungetService[p.123]

4.23.22 public interface SynchronousBundleListener extends BundleListener

A synchronous BundleEvent listener.

SynchronousBundleListener is a listener interface that may be implemented by a bundle developer.

A SynchronousBundleListener object is registered with the Framework using the BundleContext.addBundleListener[p.99] method. SynchronousBundleListener objects are called with a BundleEvent object when a bundle has been installed, started, stopped, updated, or uninstalled.

Unlike normal BundleListener objects, SynchronousBundleListeners are synchronously called during bundle life cycle processing. The bundle life cycle processing will not proceed until all SynchronousBundleListeners have completed. SynchronousBundleListener objects will be called prior to BundleListener objects.

AdminPermission is required to add or remove a SynchronousBundleListener object.

See Also BundleEvent[p.108]

Since 1.1

4.24 References

[9] *The Standard for the Format of ARPA Internet Text Messages*
STD 11, RFC 822, UDEL, August 1982

http://www.ietf.org/rfc/rfc822.txt

[10] *The Hypertext Transfer Protocol - HTTP/1.1*
RFC 2068 DEC, MIT/LCS, UC Irvine, January 1997
http://www.ietf.org/rfc/rfc2068.txt

[11] *The Java 2 Platform API Specification*
Standard Edition, Version 1.3, Sun Microsystems
http://java.sun.com/j2se/1.4

[12] *The Java Language Specification*
Second Edition, Sun Microsystems, 2000
http://java.sun.com/docs/books/jls/index.html

[13] *A String Representation of LDAP Search Filters*
RFC 1960, UMich, 1996
http://www.ietf.org/rfc/rfc1960.txt

[14] *The Java Security Architecture for JDK 1.2*
Version 1.0, Sun Microsystems, October 1998
http://java.sun.com/products/jdk/1.4/docs/guide/security/spec/security-spec.doc.html

[15] *The Java 2 Package Versioning Specification*
http://java.sun.com/j2se/1.4/docs/guide/versioning/index.html

[16] *Codes for the Representation of Names of Languages*
ISO 639, International Standards Organization
http://lcweb.loc.gov/standards/iso639-2/langhome.html

[17] *Manifest Format*
http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html#JAR%20Manifest

5 Package Admin Service Specification

Version 1.1

5.1 Introduction

Bundles can export packages to other bundles. This exporting creates a dependency between the bundle exporting a package and the bundle using the package. When the exporting bundle is uninstalled or updated, a decision must be taken regarding any shared packages.

The Package Admin service provides an interface to let the Management Agent make this decision.

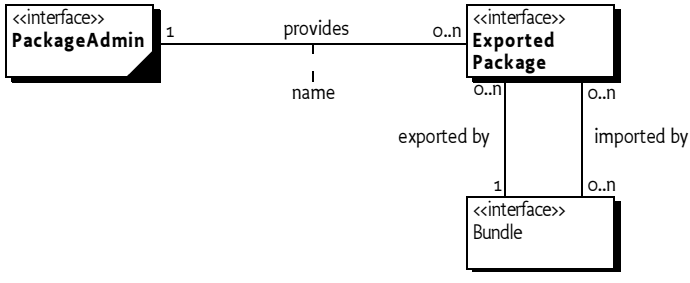
5.1.1 Essentials

- *Information* – The Package Admin service must provide the sharing status of all packages. This should include information about the importing bundles and exporting bundle.
- *Policy* – The Package Admin service must allow a management agent to provide a policy for package sharing when bundles are updated and uninstalled.
- *Minimal update* – Only bundles that depend on the package that needs to be resolved should have to be restarted when packages are forced to be refreshed.

5.1.2 Entities

- PackageAdmin – The interface that provides access to the internal Framework package sharing mechanism.
- ExportedPackage – The interface provides package information and its sharing status.
- *Management Agent* – A bundle that is provided by the Operator to implement an Operator specific policy.

Figure 20 *Class Diagram org.osgi.service.packageadmin*



5.1.3 Operation

The Framework's system bundle should provide a Package Admin service for the Management Agent. The Package Admin service must be registered under the org.osgi.service.packageadmin.PackageAdmin interface by the system bundle. It provides access to the internal structures of the Framework related to package sharing. See *Sharing Packages* on page 46. This is an optional singleton service, so at most one Package Admin service must be registered at any moment in time.

The Framework must always leave the package sharing intact for packages exported by a bundle that is uninstalled or updated. A Management Agent can then choose to force the framework to refresh these packages using the Package Admin service. A policy of always using the most current packages exported by installed bundles can be implemented with a Management Agent that watches Framework events for bundles being uninstalled or updated, and then refreshes the packages of those bundles using the Package Admin service.

5.2 Package Admin

The Package Admin service is intended to allow a Management Agent to define the policy for managing package sharing. It provides methods for examining the status of the shared packages. It also allows the Management Agent to refresh the packages and stop and restart bundles as necessary.

The PackageAdmin class provides the following methods:

- getExportedPackage(String) – Returns an ExportedPackage object that provides information about the requested package. This information can be used to make the decision to refresh the package.
- getExportedPackages(Bundle) – Returns a list of ExportedPackage objects for each package that the given bundle exports.
- refreshPackages(Bundle[]) – The management agent may call this method to refresh the exported packages of the specified bundles. The actual work must happen asynchronously. The Framework must send a Framework.PACKAGES_REFRESHED when all packages have been refreshed.

Information about the shared packages is provided by the ExportedPackage objects. These objects provide detailed information about the bundles that import and export the package. This information can be used by a Management Agent to guide its decisions.

5.3 Security

The Package Admin service is a *system service* that can easily be abused because it provides access to the internal data structures of the Framework. Many bundles may have the ServicePermission[GET, org.osgi.service.packageadmin.PackageAdmin] because AdminPermission

is required for calling any of the methods that modify the environment. No bundle must have ServicePermission[REGISTER, org.osgi.service.packageadmin.PackageAdmin], because only the Framework itself should register such a system service.

This service should only be used by a Management Agent.

5.4 Changes

- The Framework must broadcast a Framework event of type PACKAGES_REFRESHED event when the Package Admin service has finished refreshing all the packages.
- The sentences describing the use of the bundle parameter to the refreshPackages constant FrameworkEvent.ERROR were added.

5.5 org.osgi.service.packageadmin

The OSGi Package Admin service 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.packageadmin; specification-version=1.1
```

5.5.1 Summary

- *ExportedPackage* – An exported package. [p.133]
- *PackageAdmin* – Framework service which allows bundle programmers to inspect the packages exported in the Framework and eagerly update or uninstall bundles. [p.134]

5.5.2 public interface ExportedPackage

An exported package. Instances implementing this interface are created by the Package Admin service.

The information about an exported package provided by this object is valid only until the next time PackageAdmin.refreshPackages() is called. If an ExportedPackage object becomes stale (that is, the package it references has been updated or removed as a result of calling PackageAdmin.refreshPackages()), its getName() and getSpecificationVersion() continue to return their old values, isRemovalPending() returns true, and getExportingBundle() and getImportingBundles() return null.

5.5.2.1 public Bundle getExportingBundle( )

□ Returns the bundle exporting the package associated with this ExportedPackage object.

Returns The exporting bundle, or null if this ExportedPackage object has become stale.

5.5.2.2 public Bundle[] getImportingBundles( )

- Returns the resolved bundles that are currently importing the package associated with this ExportedPackage object.

  The returned array always includes the bundle returned by getExportingBundle[p.133] since an exporter always implicitly imports its exported packages.

Returns The array of resolved bundles currently importing the package associated with this ExportedPackage object, or null if this ExportedPackage object has become stale.

5.5.2.3 public String getName( )

- Returns the name of the package associated with this ExportedPackage object.

Returns The name of this ExportedPackage object.

5.5.2.4 public String getSpecificationVersion( )

- Returns the specification version of this ExportedPackage, as specified in the exporting bundle's manifest file.

Returns The specification version of this ExportedPackage object, or null if no version information is available.

5.5.2.5 public boolean isRemovalPending( )

- Returns true if the package associated with this ExportedPackage object has been exported by a bundle that has been updated or uninstalled.

Returns true if the associated package is being exported by a bundle that has been updated or uninstalled, or if this ExportedPackage object has become stale; false otherwise.

5.5.3 public interface PackageAdmin

Framework service which allows bundle programmers to inspect the packages exported in the Framework and eagerly update or uninstall bundles. If present, there will only be a single instance of this service registered with the Framework.

The term *exported package* (and the corresponding interface ExportedPackage[p.133])refers to a package that has actually been exported (as opposed to one that is available for export).

The information about exported packages returned by this service is valid only until the next time refreshPackages[p.135] is called. If an ExportedPackage object becomes stale, (that is, the package it references has been updated or removed as a result of calling PackageAdmin.refreshPackages()), its getName() and getSpecificationVersion() continue to return their old values, isRemovalPending() returns true, and getExportingBundle() and getImportingBundles() return null.

5.5.3.1 public ExportedPackage getExportedPackage(String name)

name The name of the exported package to be returned.

- Gets the ExportedPackage object with the specified package name. All exported packages will be checked for the specified name. In an environment where the exhaustive list of packages on the system classpath is not known in advance, this method attempts to see if the named package is on the system classpath. This means that this method may discover an ExportedPackage object that was not present in the list returned by a prior call to getExportedPackages().

Returns The exported package with the specified name, or null if no expored package with that name exists.

5.5.3.2 public ExportedPackage[] getExportedPackages(Bundle bundle)

bundle The bundle whose exported packages are to be returned, or null if all the packages currently exported in the Framework are to be returned. If the specified bundle is the system bundle (that is, the bundle with id zero), this method returns all the packages on the system classpath whose name does not start with "java.". In an environment where the exhaustive list of packages on the system classpath is not known in advance, this method will return all currently known packages on the system classpath, that is, all packages on the system classpath that contains one or more classes that have been loaded.

- Gets the packages exported by the specified bundle.

Returns The array of packages exported by the specified bundle, or null if the specified bundle has not exported any packages.

5.5.3.3 public void refreshPackages(Bundle[] bundles)

bundles the bundles whose exported packages are to be updated or removed, or null for all previously updated or uninstalled bundles.

- Forces the update (replacement) or removal of packages exported by the specified bundles.

  If no bundles are specified, this method will update or remove any packages exported by any bundles that were previously updated or uninstalled since the last call to this method. The technique by which this is accomplished may vary among different Framework implementations. One permissible implementation is to stop and restart the Framework.

  This method returns to the caller immediately and then performs the following steps in its own thread:

  1 Compute a graph of bundles starting with the specified bundles. If no bundles are specified, compute a graph of bundles starting with previously updated or uninstalled ones. Add to the graph any bundle that imports a package that is currently exported by a bundle in the graph. The graph is fully constructed when there is no bundle outside the graph that imports a package from a bundle in the graph. The graph may contain UNINSTALLED bundles that are currently still exporting packages.
  2 Each bundle in the graph that is in the ACTIVE state will be stopped as described in the Bundle.stop method.
  3 Each bundle in the graph that is in the RESOLVED state is moved to the INSTALLED state. The effect of this step is that bundles in the graph are no longer RESOLVED.
  4 Each bundle in the graph that is in the UNINSTALLED state is removed from the graph and is now completely removed from the Framework.

5 Each bundle in the graph that was in the ACTIVE state prior to Step 2 is started as described in the Bundle.start method, causing all bundles required for the restart to be resolved. It is possible that, as a result of the previous steps, packages that were previously exported no longer are. Therefore, some bundles may be unresolvable until another bundle offering a compatible package for export has been installed in the Framework.
6 A framework event of type FrameworkEvent.PACKAGES_REFRESHED is broadcast.

For any exceptions that are thrown during any of these steps, a FrameworkEvent of type ERROR is broadcast, containing the exception. The source bundle for these events should be the specific bundle to which the exception is related. If no specific bundle can be associated with the exception then the System Bundle must be used as the source bundle for the event.

Throws SecurityException – if the caller does not have the AdminPermission and the Java runtime environment supports permissions.

6 Start Level Service Specification

Version 1.0

6.1 Introduction

This specification describes how to enable a Management Agent to control the relative starting and stopping order of bundles in an OSGi Service Platform.

The Start Level service assigns each bundle a *start level*. The Management Agent can modify the start levels for bundles and set the active start level of the Framework, which will start and stop the appropriate bundles. Only bundles that have a start level less or equal to this active start level must be active.

The purpose of the Start Level service is to allow the Management Agent to control, in detail, what bundles get started and stopped and when this occurs.

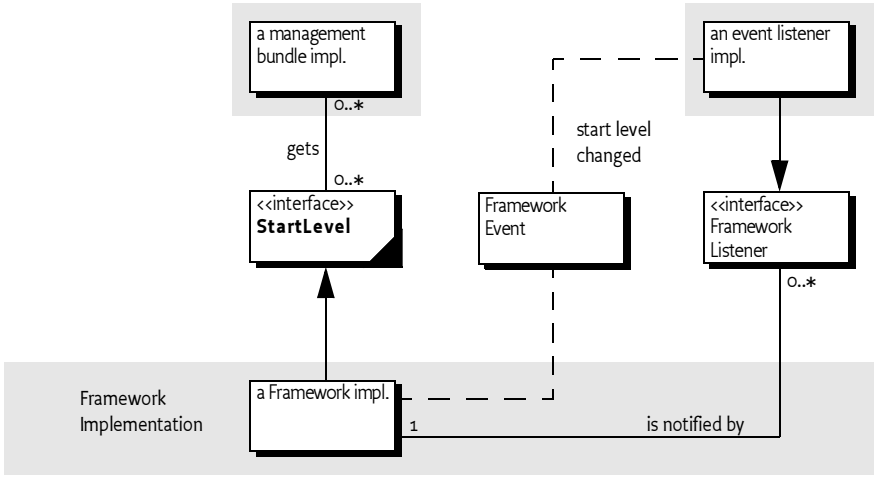
6.1.1 Essentials

- *Ordering* – A management agent should be able to order the startup and shutdown sequences of bundles.
- *Levels* – The management agent should support a virtually unlimited number of levels.
- *Backward compatible* – The model for start levels should be compatible with the OSGi Service Platform Release 2 specifications.

6.1.2 Entities

- *Start Level Service* – The service that is used by a Management Agent to order the startup and shutdown sequences of bundles.
- *Management Agent* – See page 32.
- *Framework Event* – See page 117.
- *Framework Listener* – See page 119.

Figure 21 Class Diagram org.osgi.service.startlevel package



6.2 Start Level Service

The Start Level Service provides the following functions:

- Controls the beginning start level of the OSGi Framework.
- Is used to modify the active start level of the Framework.
- Can be used to assign a specific start level to a bundle.
- Can set the initial start level for newly installed bundles.

Defining the order in which bundles are started and stopped is useful for the following:

- *Safe mode* – The Management Agent can implement a *safe mode*. In this mode, only fully trusted bundles are started. Safe mode might be necessary when a bundle causes a failure at startup that disrupts normal operation and prevents correction of the problem.
- *Splash screen* – If the total startup time is long, it might be desirable to show a splash screen during initialization. This improves the user's perception of the boot time of the device. The startup ordering can ensure that the right bundle is started first.
- *Handling erratic bundles* – Problems can occur because bundles require services to be available when they get activated (this is a programming error). By controlling the start order, the Management Agent can prevent these problems.
- *High priority bundles* – Certain tasks such as metering need to run as quickly as possible and cannot have a long startup delay. These bundles can be started first.

6.2.1 The Concept of a Start Level

A *start level* is defined as a non-negative integer. A start level of 0 (zero) is the state in which the Framework has either not been launched or has completed shutdown (these two states are considered equivalent). In this state, no bundles are running. Progressively higher integral values represent progressively higher start levels. For example, 2 is a higher start level than 1. The Framework must support all positive int values (Integer.MAX_VALUE) for start levels.

The Framework has an *active start level* that is used to decide which bundles can be started. All bundles must be assigned a *bundle start level.* This is the minimum start level for which a bundle can be started. The bundle start level can be set with the setBundleStartLevel(Bundle,int) method. When a bundle is installed, it is intially assigned the bundle start level returned by getInitialBundleStartLevel(). The initial bundle start level to be used when bundles are installed can be set with setInitialBundleStartLevel(int).

Additionally, a bundle can be persistently marked as *started* or *stopped* with the Bundle start and stop methods. A bundle cannot run unless it is marked started, regardless of the bundle's start level.

6.2.2 Changing the Active Start Level

A Management Agent can influence the active start level with the setStartLevel(int) method. The Framework must then step-wise increase or decrease the active start level until the requested start level is reached. The process of starting or stopping bundles, which is initiated by the setStartLevel(int) method, must take place asynchronously.

This means that the *active start level* (the one that is active at a certain moment in time) must be changed to a new start level, called the *requested start level.* The active and requested levels differ during a certain period when the Framework starts and stops the appropriate bundles. Moving from the active start level to the requested start level must take place in increments of one (1).

If the requested start level is higher than the active start level, the Framework must increase the start level by one and then start all bundles, that meet the following criteria:

- Bundles that are persistently marked started, and
- have a bundle start level equal to the new active start level.

The Framework continues increasing the active start level and starting the appropriate bundles until it has started all bundles with a bundle start level that equals the requested start level.

The Framework must not increase to the next active start level until all started bundles have returned from their BundleActivator.start method normally or with an exception. A FrameworkEvent.ERROR must be broadcast when the BundleActivator.start method throws an exception.

If the requested start level is lower than the active start level, the Framework must stop all bundles that have a bundle start level that is equal to the active start level. The Framework must then decrease the active start level by 1. If the active start level is still higher than the requested start level, it should

continue stopping the appropriate bundles and decreasing the active start level until the requested start level is reached. A FrameworkEvent.ERROR must be broadcast when the BundleActivator.stop method throws an exception.

If the requested start level is the active start level, the Framework will not start or stop any bundles.

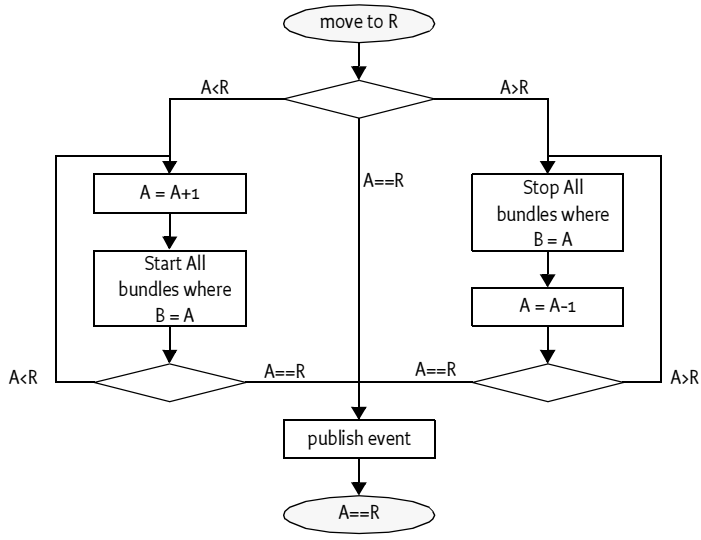
When the requested start level is reached and all bundles satisfy the condition that their bundle start level <= active start level in order to be started, then the FrameworkEvent.STARTLEVEL_CHANGED event must be sent to all registered FrameworkListener objects. If the requested start level and active start level are equal, then this event may arrive before the setStartLevel method has returned.

It must therefore always be true that:

- A bundle is started, or will be started in a short period of time, if the start level is less or equal to the active start level.
- A bundle is stopped, or will be stopped soon, when it has a start level more than the active start level.

These steps are depicted in the flow chart in Figure 22.

Figure 22 *Move to requested start level R, active level is A, B is a bundle's start level*



If the Framework is currently involved in changing the active start level, it must first reach the previously requested start level before it is allowed to continue with a newly requested start level. For example, assume the active start level is 5 and the Framework is requested to transition to start level 3. Before start level 3 is reached, another request is made to transition to start level 7. In this case, the OSGi Framework must first complete the transition to start level 3 before it transitions to start level 7.

6.2.3 Startup sequence

At startup, the Framework must have an active start level of zero. It must then move the active start level to the *beginning start level*. The beginning start level is specified with an argument when starting the Framework or through some other means, which is left undefined here. If no beginning start level is given, the Framework must assume a beginning start level of one (1).

The Framework must launch and then set the requested start level to the beginning start level. It must then follow the procedure described in *Changing the Active Start Level* on page 139 to make the active start level equal the beginning start level, with the exception of the FrameworkEvent.START_LEVEL_CHANGED event broadcast. During launching, the Framework must broadcast a FrameworkEvent.STARTED event when the initial start level is reached.

6.2.4 Shutdown Sequence

When the Framework shuts down, the requested start level must be set to zero. The Framework must then follow the process described in *Changing the Active Start Level* on page 139 to make the active start level equal to zero.

6.2.5 Changing a Bundle's Start Level

Bundles are assigned an initial start level when they are installed. The default value for the initial start level is set to one, but can be changed with the setInitialBundleStartLevel(int) method. A bundle's start level will not change when the setInitialBundleStartLevel(int) method later modifies the default initial start level.

Once installed, the start level of a bundle can be changed with setBundleStartLevel(Bundle,int). When a bundle's start level is changed and the bundle is marked persistently to be started, then the OSGi Framework must compare the new bundle start level to the active start level. For example, assume that the active start level is 5 and a bundle with start level 5 is started. If the bundle's start level subsequently is changed to 6 then this bundle must be stopped by the OSGi Framework but it must still be marked persistently to be started.

6.2.6 Starting a Bundle

If a bundle is started by calling the Bundle.start() method, then the OSGi Framework must mark the bundle as persistently started. The OSGi Framework must not actually start a bundle when the active start level is less than the bundle's start level. In that case, the state must not change.

6.2.7 Exceptions in the Bundle Activator

If the BundleActivator.start or stop method throws an Exception, then the handling of this Exception is different depending who invoked the start or stop method.

If the bundle gets started/stopped due to a change in the active start level, then the Exception must be broadcast as a FrameworkEvent.ERROR event. Otherwise a new BundleException is thrown to the caller.

6.2.8 System Bundle

The System Bundle is defined to have a start level of zero. See page 42 for more information on the System Bundle start level. The start level of the System Bundle cannot be changed. An IllegalArgumentException must be thrown if an attempt is made to change the start level of the System Bundle.

6.3 Compatibility Mode

Compatibility mode consists of a single start level for all bundles. All bundles are assigned a bundle start level of 1. In compatibility mode, the OSGi Framework is started and launched with an argument specifying an beginning start level of 1. The Framework then starts all bundles that are persistently marked to be started. When start level 1 is reached, all bundles have been started and the FrameworkEvent.STARTED event is published. This is considered compatible with prior OSGi Framework versions because all bundles are started and there is no control over the start order. Framework implementations must support compatibility mode.

6.4 Example Applications

The Start Level service allows a Management Agent to implement many different startup schemes. The following sections show some examples.

6.4.1 Safe Mode Startup Scheme

A Management Agent can implement a *safe mode* in which it runs trusted bundles at level 1 and runs itself on level 2. When the Management Agent gets control, it constructs a list of all applications to be started. This list can be constructed from BundleContext.getBundles(). The Management Agent checks each bundle to determine if it is not started but is marked to be started persistently by calling the isBundlePersistentlyStarted(Bundle) method of the Start Level service.

Before it starts each bundle, the Management Agent persistently records the bundle to be started and then starts the bundle. This continues until all bundles are started. When all bundles are successfully started, the Management Agent persistently records that all bundles started without problems.

If the Service Platform is restarted, the Management Agent should inspect the persistently recorded information. If the persistently recorded information indicates a bundle failure, the Management Agent should try to restart the system without that application bundle since that bundle failed. Alternatively, it could contact its Remote Manager and ask for assistance.

6.4.2 Splash Screen Startup Scheme

A splash screen is a popup containing startup information about an application. The popup provides feedback to the end user indicating that the system is still initializing. The Start Level service can be used by a bundle to pop-up a splash screen before any other bundle is started, and remove it once all bundles have been started. The splash-screen bundle would start at start level 1 and all other bundles would start at start level 2 or higher.

```
class SplashScreen implements
   BundleActivator, FrameworkListener {
   Screen   screen;
   public void start(BundleContext context) {
      context.addFrameworkListener( this );
      screen = createSplash();
      screen.open();
   }
   public void stop(BundleContext context) {
      screen.close();
   }
   public void frameworkEvent( FrameworkEvent event ) {
      if ( event.getType() == FrameworkEvent.STARTED )
         screen.close();
   }
   Screen createSplash() { ... }
}
```

6.5 Security

When the Start Level service is available, it is crucial to protect its usage from non-trusted bundles. A malicious bundle that can control start levels can control the whole service platform.

The Start Level service is intended to be used only by a Management Agent. This means that bundles that use this service must have AdminPermission to be able to modify a bundle's start level or the Framework's active start level. Bundles that need only read access to this service should have ServicePermission[GET,StartLevel].

The Start Level service must be registered by the Framework so there is no reason for any bundle to have ServicePermission[REGISTER,StartLevel].

6.6 org.osgi.service.startlevel

The OSGi StartLevel service 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.startlevel; specification-
version=1.0
```

6.6.1 public interface StartLevel

The StartLevel service allows management agents to manage a start level assigned to each bundle and the active start level of the Framework. There is at most one StartLevel service present in the OSGi environment.

A start level is defined to be a state of execution in which the Framework exists. StartLevel values are defined as unsigned integers with 0 (zero) being the state where the Framework is not launched. Progressively higher integral values represent progressively higher start levels. e.g. 2 is a higher start level than 1.

Access to the StartLevel service is protected by corresponding `ServicePermission`. In addition the `AdminPermission` that is required to actually modify start level information.

Start Level support in the Framework includes the ability to control the beginning start level of the Framework, to modify the active start level of the Framework and to assign a specific start level to a bundle. How the beginning start level of a Framework is specified is implementation dependent. It may be a command line argument when invoking the Framework implementation.

When the Framework is first started it must be at start level zero. In this state, no bundles are running. This is the initial state of the Framework before it is launched. When the Framework is launched, the Framework will enter start level one and all bundles which are assigned to start level one and are persistently marked to be started are started as described in the `Bundle.start` method. Within a start level, bundles are started in ascending order by `Bundle.getBundleId`. The Framework will continue to increase the start level, starting bundles at each start level, until the Framework has reached a beginning start level. At this point the Framework has completed starting bundles and will then broadcast a Framework event of type `FrameworkEvent.STARTED` to announce it has completed its launch.

The StartLevel service can be used by management bundles to alter the active start level of the framework.

6.6.1.1 public int getBundleStartLevel(Bundle bundle)

bundle The target bundle.

□ Return the assigned start level value for the specified Bundle.

Returns The start level value of the specified Bundle.

Throws `IllegalArgumentException` – If the specified bundle has been uninstalled.

6.6.1.2 public int getInitialBundleStartLevel( )

□ Return the initial start level value that is assigned to a Bundle when it is first installed.

Returns The initial start level value for Bundles.

See Also `setInitialBundleStartLevel`[p.145]

6.6.1.3 public int getStartLevel( )

□ Return the active start level value of the Framework. If the Framework is in the process of changing the start level this method must return the active start level if this differs from the requested start level.

Returns The active start level value of the Framework.

6.6.1.4 public boolean isBundlePersistentlyStarted(Bundle bundle)

□ Return the persistent state of the specified bundle.

This method returns the persistent state of a bundle. The persistent state of a bundle indicates whether a bundle is persistently marked to be started when it's start level is reached.

Returns `true` if the bundle is persistently marked to be started, `false` if the bundle is not persistently marked to be started.

Throws `IllegalArgumentException` – If the specified bundle has been uninstalled.

6.6.1.5 public void setBundleStartLevel(Bundle bundle, int startlevel)

bundle The target bundle.

startlevel The new start level for the specified Bundle.

- Assign a start level value to the specified Bundle.

  The specified bundle will be assigned the specified start level. The start level value assigned to the bundle will be persistently recorded by the Framework. If the new start level for the bundle is lower than or equal to the active start level of the Framework, the Framework will start the specified bundle as described in the `Bundle.start` method if the bundle is persistently marked to be started. The actual starting of this bundle must occur asynchronously. If the new start level for the bundle is higher than the active start level of the Framework, the Framework will stop the specified bundle as described in the `Bundle.stop` method except that the persistently recorded state for the bundle indicates that the bundle must be restarted in the future. The actual stopping of this bundle must occur asynchronously.

Throws `IllegalArgumentException` – If the specified bundle has been uninstalled or if the specified start level is less than or equal to zero, or the specified bundle is the system bundle.

`SecurityException` – if the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

6.6.1.6 public void setInitialBundleStartLevel(int startlevel)

startlevel The initial start level for newly installed bundles.

- Set the initial start level value that is assigned to a Bundle when it is first installed.

  The initial bundle start level will be set to the specified start level. The initial bundle start level value will be persistently recorded by the Framework.

  When a Bundle is installed via `BundleContext.installBundle`, it is assigned the initial bundle start level value.

  The default initial bundle start level value is `1` unless this method has been called to assign a different initial bundle start level value.

  Thie method does not change the start level values of installed bundles.

Throws `IllegalArgumentException` – If the specified start level is less than or equal to zero.

`SecurityException` – if the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

6.6.1.7 public void setStartLevel(int startlevel)

startlevel The requested start level for the Framework.

- Modify the active start level of the Framework.

The Framework will move to the requested start level. This method will return immediately to the caller and the start level change will occur asynchronously on another thread.

If the specified start level is higher than the active start level, the Framework will continue to increase the start level until the Framework has reached the specified start level, starting bundles at each start level which are persistently marked to be started as described in the `Bundle.start` method. At each intermediate start level value on the way to and including the target start level, the framework must:

1 Change the active start level to the intermediate start level value.
2 Start bundles at the intermediate start level in ascending order by `Bundle.getBundleId`.

`FrameworkEvent.STARTLEVEL_CHANGED` to announce it has moved to the specified start level.

If the specified start level is lower than the active start level, the Framework will continue to decrease the start level until the Framework has reached the specified start level stopping bundles at each start level as described in the `Bundle.stop` method except that their persistently recorded state indicates that they must be restarted in the future. At each intermediate start level value on the way to and including the specified start level, the framework must:

1 Stop bundles at the intermediate start level in descending order by `Bundle.getBundleId`.
2 Change the active start level to the intermediate start level value.

`FrameworkEvent.STARTLEVEL_CHANGED` to announce it has moved to the specified start level.

If the specified start level is equal to the active start level, then no bundles are started or stopped, however, the Framework must broadcast a Framework event of type `FrameworkEvent.STARTLEVEL_CHANGED` to announce it has finished moving to the specified start level. This event may arrive before the this method return.

Throws `IllegalArgumentException` – If the specified start level is less than or equal to zero.

`SecurityException` – If the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

7 Permission Admin Service Specification

Version 1.1

7.1 Introduction

In the Framework, a bundle can have a single set of permissions. These permissions are used to verify that a bundle is authorized to execute privileged code. For example, a FilePermission defines what files can be used and in what way.

The policy of providing the permissions to the bundle should be delegated to a Management Agent. For this reason, the Framework provides the Permission Admin service so that a Management Agent can administrate the permissions of a bundle and provide defaults for all bundles.

Related mechanisms of the Framework are discussed in *Security* on page 80.

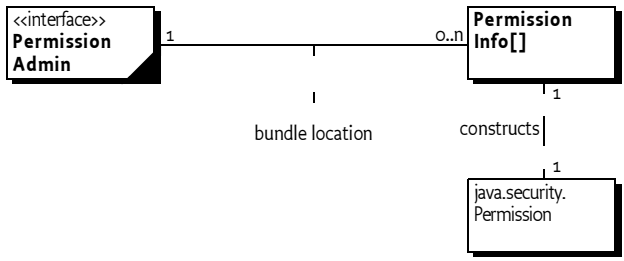
7.1.1 Essentials

- *Status information* – The Permission Admin Service must provide status information about the current permissions of a bundle.
- *Administrative* – The Permission Admin Service must allow a Management Agent to set the permissions before, during, or after a bundle is installed.
- *Defaults* – The Permission Admin Service must provide control over default permissions. These are the permissions for a bundle with no specific permissions set.

7.1.2 Entities

- PermissionAdmin – The service that provides access to the permission repository of the Framework.
- PermissionInfo – An object that holds the information needed to construct a Permission object.
- *Bundle location* – The string that specifies the bundle location. This is described in *Bundle Location* on page 57.

Figure 23 *Class Diagram org.osgi.service.permissionadmin.*



7.1.3 Operation

The Framework maintains a repository of permissions. These permissions are stored under the bundle location string. Using the bundle location allows the permissions to be set *before* a bundle is downloaded. The Framework must consult this repository when it needs the permissions of a bundle. When no specific permissions are set, the bundle must use the default permissions. If no default is set, the bundle must use java.security.AllPermission. If the default permissions are changed, a bundle with no specific permissions must immediately start using the new default permissions.

The Permission Admin service is registered by the Framework's system bundle under the org.osgi.service.permissionadmin.PermissionAdmin interface. This is an optional singleton service, so at most one Permission Admin service is registered at any moment in time.

The Permission Admin service provides access to the permission repository. A Management Agent can get, set, update, and delete permissions from this repository. A Management Agent can also use a SynchronousBundleListener object to set the permissions during the installation or updating of a bundle.

7.2 Permission Admin service

The Permission Admin service needs to manipulate the default permissions and the permissions associated with a specific bundle. The default permissions and the bundle-specific permissions are stored persistently. It is possible to set a bundle's permissions before the bundle is installed in the Framework because the bundle's location is used to set the bundle's permissions.

The manipulation of a bundle's permissions, however, may also be done in real time when a bundle is downloaded or just before the bundle is downloaded. To support this flexibility, a SynchronousBundleListener object may be used by a Management Agent to detect the installation or update of a bundle, and set the required permissions before the installation completes.

Permissions are activated before the first time a permission check for a bundle is performed. This means that if a bundle has opened a file, this file must remain usable even if the permission to open that file is removed at a later time.

Permission information is *not* specified using java.security.Permission objects. The reason for this approach is the relationship between the required persistence of the information across Framework restarts and the concept of classloaders in the Framework. Actual Permission classes must be subclasses of Permission and may be exported from any bundle. The Framework can access these permissions as long as they are exported, but the Management Agent would have to import all possible packages that contain permissions. This requirement would severely limit permission types. Therefore, the Permission Admin service uses the PermissionInfo class to specify permission information. Objects of this class are used by the Framework to create Permission objects.

PermissionInfo objects restrict the possible Permission objects that can be used. A Permission subclass can only be described by a PermissionInfo object when it has the following characteristics:

- It must be a subclass of java.security.Permission.
- It must use the two-argument public constructor type(name,actions).
- The class must be available to the Framework code from the system classpath or from any exported package so it can be loaded by the Framework.
- The class must be public.

If any of these conditions is not met, the PermissionInfo object must be ignored and an error message should be logged.

The permissions are always set as an array of PermissionInfo objects to make the assignment of all permissions atomic.

The PermissionAdmin interface provides the following methods:

- getLocations() – Returns a list of locations that have permissions assigned to them. This method allows a Management Agent to examine the current set of permissions.
- getPermissions(String) – Returns a list of PermissionInfo objects that are set for that location, or returns null if no permissions are set.
- setPermissions(String,PermissionInfo[]) – Associates permissions with a specific location, or returns null when the permissions should be removed.
- getDefaultPermissions() – This method returns the list of default permissions.
- setDefaultPermissions(PermissionInfo[]) – This method sets the default permissions.

7.2.1 FilePermission for Relative Path Names

A java.io.FilePermission assigned to a bundle via the setPermissions method must receive special treatment if the path argument for the FilePermission is a relative path name. A relative path name is one that is not absolute. See the java.io.File.isAbsolute method for more information on absolute path names.

When a bundle is assigned a FilePermission for a relative path name, the path name is taken to be relative to the bundle's persistent storage area. This allows additional permissions, such as "execute", to be assigned to files in the bundle's persistent storage area. For example:

```
java.io.FilePermission "-" "execute"
```

can be used to allow a bundle to execute any file in the bundle's persistent storage area.

This only applies to FilePermission objects assigned to a bundle via the setPermission method. This does not apply to default permissions. A FilePermission for a relative path name assigned via the setDefaultPermission method must be ignored.

7.3 Security

The Permission Admin service is a system service that can be abused. A bundle that can access and use the Permission Admin service has full control over the OSGi Service Platform. However, many bundles can have ServicePermission[GET,PermissionAdmin] because all methods that change the state of the Framework require AdminPermission.

No bundle must have ServicePermission[REGISTER,PermissionAdmin] for this service because only the Framework should provide this service.

7.4 Changes

The following descriptions were added relative to the previous version of this specification:

- A section was added to this specification that defines how the names of FilePermission objects should be treated.
- NullPointerException and IllegalArgumentException were added to PermissionInfo(String, String, String).
- Clarification to PermissionInfo.getEncoded about whitespace was added.
- The documentation of the PermissionAdmin.getDefaultPermissions method was updated to avoid using "not defined".
- Documentation to the PermissionAdmin.setDefaultPermissions method regarding a null argument was added.

7.5 org.osgi.service.permissionadmin

The OSGi Permission Admin service 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.permissionadmin; specification-version=1.1
```

7.5.1 Summary

- *PermissionAdmin* – The Permission Admin service allows management agents to manage the permissions of bundles. [p.150]
- *PermissionInfo* – Permission representation used by the Permission Admin service. [p.152]

7.5.2 public interface PermissionAdmin

The Permission Admin service allows management agents to manage the permissions of bundles. There is at most one Permission Admin service present in the OSGi environment.

Access to the Permission Admin service is protected by corresponding `ServicePermission`. In addition `AdminPermission` is required to actually set permissions.

Bundle permissions are managed using a permission table. A bundle's location serves as the key into this permission table. The value of a table entry is the set of permissions (of type `PermissionInfo`) granted to the bundle named by the given location. A bundle may have an entry in the permission table prior to being installed in the Framework.

The permissions specified in `setDefaultPermissions` are used as the default permissions which are granted to all bundles that do not have an entry in the permission table.

Any changes to a bundle's permissions in the permission table will take effect no later than when bundle's `java.security.ProtectionDomain` is next involved in a permission check, and will be made persistent.

Only permission classes on the system classpath or from an exported package are considered during a permission check. Additionally, only permission classes that are subclasses of `java.security.Permission` and define a 2-argument constructor that takes a *name* string and an *actions* string can be used.

Permissions implicitly granted by the Framework (for example, a bundle's permission to access its persistent storage area) cannot be changed, and are not reflected in the permissions returned by `getPermissions` and `getDefaultPermissions`.

7.5.2.1 public PermissionInfo[] getDefaultPermissions( )

- Gets the default permissions.

  These are the permissions granted to any bundle that does not have permissions assigned to its location.

Returns The default permissions, or `null` if no default permissions are set.

7.5.2.2 public String[] getLocations( )

- Returns the bundle locations that have permissions assigned to them, that is, bundle locations for which an entry exists in the permission table.

Returns The locations of bundles that have been assigned any permissions, or `null` if the permission table is empty.

7.5.2.3 public PermissionInfo[] getPermissions(String location)

location The location of the bundle whose permissions are to be returned.

- Gets the permissions assigned to the bundle with the specified location.

Returns The permissions assigned to the bundle with the specified location, or `null` if that bundle has not been assigned any permissions.

7.5.2.4 public void setDefaultPermissions(PermissionInfo[] permissions)

permissions The default permissions, or `null` if the default permissions are to be removed from the permission table.

- Sets the default permissions.

  These are the permissions granted to any bundle that does not have permissions assigned to its location.

Throws `SecurityException` – if the caller does not have the `AdminPermission`.

7.5.2.5 public void setPermissions(String location, PermissionInfo[] permissions)

location The location of the bundle that will be assigned the permissions.

permissions The permissions to be assigned, or `null` if the specified location is to be removed from the permission table.

- Assigns the specified permissions to the bundle with the specified location.

Throws `SecurityException` – if the caller does not have the `AdminPermission`.

7.5.3 public class PermissionInfo

Permission representation used by the Permission Admin service.

This class encapsulates three pieces of information: a Permission *type* (class name), which must be a subclass of `java.security.Permission`, and the *name* and *actions* arguments passed to its constructor.

In order for a permission represented by a `PermissionInfo` to be instantiated and considered during a permission check, its Permission class must be available from the system classpath or an exported package. This means that the instantiation of a permission represented by a `PermissionInfo` may be delayed until the package containing its Permission class has been exported by a bundle.

7.5.3.1 public PermissionInfo(String type, String name, String actions)

type The fully qualified class name of the permission represented by this `PermissionInfo`. The class must be a subclass of `java.security.Permission` and must define a 2-argument constructor that takes a *name* string and an *actions* string.

name The permission name that will be passed as the first argument to the constructor of the `Permission` class identified by `type`.

actions The permission actions that will be passed as the second argument to the constructor of the `Permission` class identified by `type`.

- Constructs a `PermissionInfo` from the given type, name, and actions.

Throws `NullPointerException` – if `type` is `null`.

`IllegalArgumentException` – if `action` is not `null` and `name` is `null`.

7.5.3.2 public PermissionInfo(String encodedPermission)

encodedPermission The encoded `PermissionInfo`.

- Constructs a `PermissionInfo` object from the given encoded `PermissionInfo` string.

Throws `IllegalArgumentException` – if `encodedPermission` is not properly formatted.

See Also `getEncoded`[p.153]

7.5.3.3 public boolean equals(Object obj)

obj The object to test for equality with this `PermissionInfo` object.

Determines the equality of two `PermissionInfo` objects. This method checks that specified object has the same type, name and actions as this `PermissionInfo` object.

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

7.5.3.4 public final String getActions( )

Returns the actions of the permission represented by this `PermissionInfo`.

Returns The actions of the permission represented by this `PermissionInfo`, or `null` if the permission does not have any actions associated with it.

7.5.3.5 public final String getEncoded( )

Returns the string encoding of this `PermissionInfo` in a form suitable for restoring this `PermissionInfo`.

The encoding format is:

`(type)`

or

`(type "`*name*`")`

or

`(type "`*name*`" "`*actions*`")`

where *name* and *actions* are strings that are encoded for proper parsing. Specifically, the ", \, carriage return, and linefeed characters are escaped using `\"`, `\\`, `\r`, and `\n`, respectively.

The encoded string must contain no leading or trailing whitespace characters. A single space character must be used between type and "*name*" and between "*name*" and "*actions*".

Returns The string encoding of this `PermissionInfo`.

7.5.3.6 public final String getName( )

Returns the name of the permission represented by this `PermissionInfo`.

Returns The name of the permission represented by this `PermissionInfo`, or `null` if the permission does not have a name.

7.5.3.7 public final String getType( )

Returns the fully qualified class name of the permission represented by this `PermissionInfo`.

Returns The fully qualified class name of the permission represented by this `PermissionInfo`.

7.5.3.8 public int hashCode( )

- Returns the hash code value for this object.

Returns A hash code value for this object.

7.5.3.9 public String toString( )

- Returns the string representation of this `PermissionInfo`. The string is created by calling the `getEncoded` method on this `PermissionInfo`.

Returns The string representation of this `PermissionInfo`.

8 URL Handlers Service Specification

Version 1.0

8.1 Introduction

This specification defines how to register new URL schemes and how to convert content-typed java.io.InputStream objects to specific Java objects.

This specification standardizes the mechanism to extend the Java run-time with new URL schemes and content handlers through bundles. Dynamically extending the URL schemes that are supported in an OSGi Service Platform is a powerful concept.

This specification is necessary because the standard Java mechanisms for extending the URL class with new schemes and different content types is not compatible with the dynamic aspects of an OSGi Service Platform. The registration of a new scheme or content type is a one time only action in Java, and once registered, a scheme or content type can never be revoked. This singleton approach to registration makes the provided mechanism impossible to use by different, independent bundles. Therefore, it is necessary for OSGi Framework implementations to hide this mechanism and provide an alternative mechanism that can be used.

The OSGi Service Platform, Release 3 specifications has also standardized a Connector service that has similar capabilities. See the *IO Connector Service Specification* on page 277.

8.1.1 Essentials

- *Multiple Access* – Multiple bundles should be allowed to register ContentHandler objects and URLStreamHandler objects.
- *Existing Schemes Availability* – Existing schemes in an OSGi Service Platform should not be overridden.
- *life-cycle Monitored* – The life-cycle of bundles must be supported. Scheme handlers and content type handlers must become unavailable when the registering bundle is stopped.
- *Simplicity* – Minimal effort should be required for a bundle to provide a new URL scheme or content type handler.

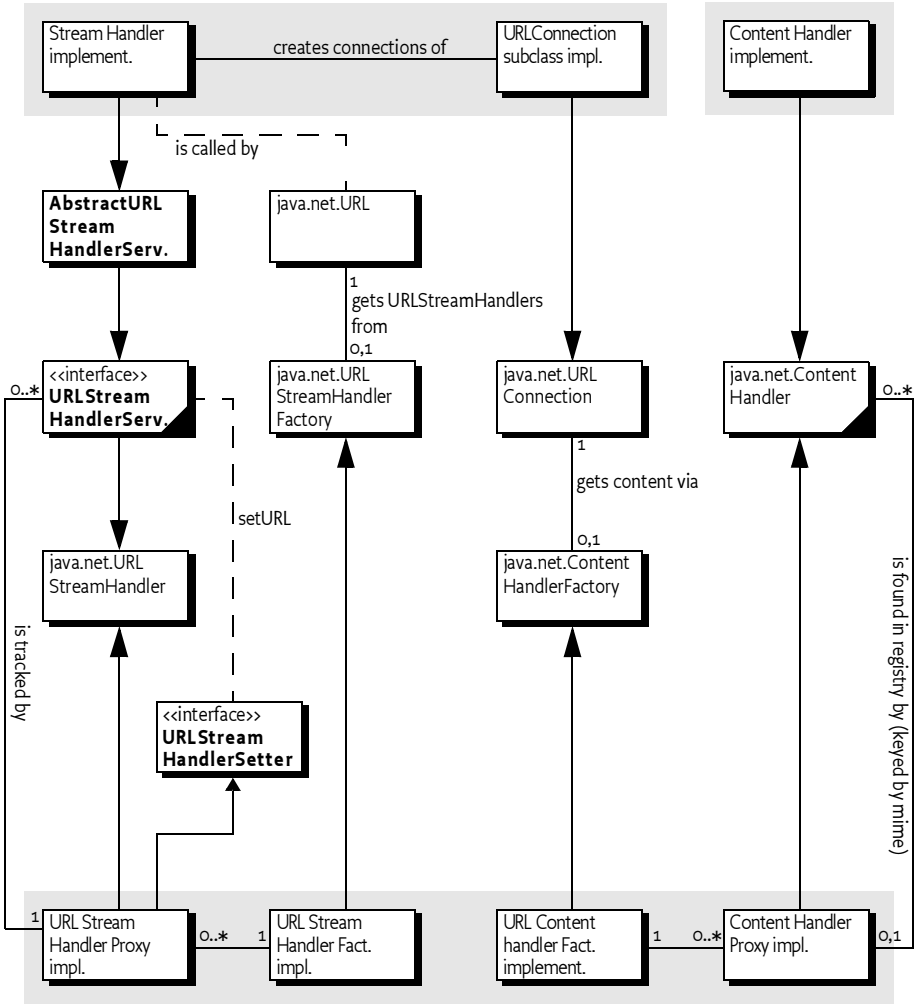
8.1.2 Entities

- *Scheme* – An identifier for a specific protocol. For example, "http" is a scheme for the Hyper Text Transfer Protocol. A scheme is implemented in a java.net.URLStreamHandler sub-class.

- *Content Type* – An identifier for the type of the content. Content types are usually referred to as MIME types. A content type handler is implemented as a java.net.ContentHandler sub-class.
- *Uniform Resource Locator (URL)* – An instance of the java.net.URL class that holds the name of a scheme with enough parameters to identify a resource for that scheme.
- *Factory* – An object that creates other objects. The purpose is to hide the implementation types (that may vary) from the caller. The created objects are a subclass/implementation of a specific type.
- *Proxy* – The object that is registered with Java and that forwards all calls to the real implementation that is registered with the service registry.
- *java.net.URLStreamHandler* – An instance of the java.net.URLStreamHandler class that can create URLConnection objects that represent a connection for a specific protocol.
- *Singleton Operation* – An operation that can only be executed once.
- *URLStreamHandlerService* – An OSGi service interface that contains the methods of the URLStreamHandler class with public visibility so they can be called from the Framework.
- *AbstractURLStreamHandlerService* – An implementation of the URLStreamHandlerService interface that implements the interface's methods by calling the implementation of the super class (java.net.url.URLStreamHandler). This class also handles the setting of the java.net.URL object via the java.net.URLStreamHandlerSetter interface.
- *URLStreamHandlerSetter* – An interface needed to abstract the setting of the java.net.URL object. This interface is related to the use of a proxy and security checking.
- *java.net.URLStreamHandlerFactory* – A factory, registered with the java.net.URL class, that is used to find java.net.URLStreamHandler objects implementing schemes that are not implemented by the Java environment. Only one java.net.URLStreamHandlerFactory object can be registered with Java.
- *java.net.URLConnection* – A connection for a specific, scheme-based protocol. A java.net.URLConnection object is created by a java.net.URLStreamHandler object when the java.net.URL.openConnection method is invoked.
- *java.net.ContentHandler* – An object that can convert a stream of bytes to a Java object. The class of this Java object depends on the MIME type of the byte stream.
- *java.net.ContentHandlerFactory* – A factory that can extend the set of java.net.ContentHandler objects provided by the java.net.URLConnection class, by creating new ones on demand. Only one java.net.ContentHandlerFactory object can be registered with the java.net.URLConnection class.
- *MIME Type* – A name-space for byte stream formats. See [20] *MIME Multipurpose Internet Mail Extension.*

The following class diagram is surprisingly complex due to the complicated strategy that Java uses to implement extendable stream handlers and content handlers.

Figure 24 Class Diagram, java.net (URL and associated classes)



8.1.3 Operation

A bundle that can implement a new URL scheme should register a service object under the URLStreamHandlerService interface with the OSGi Framework. This interface contains public versions of the java.net.URLStreamHandler class methods, so that these methods can be called by the *proxy* (the object that is actually registered with the Java runtime).

The OSGi Framework implementation must make this service object available to the underlying java.net implementation. This must be supported by the OSGi Framework implementation because the java.net.URL.setStreamHandlerFactory method can only be called *once*, making it impossible to use by bundles that come and go.

Bundles that can convert a content-typed stream should register a service object under the name java.net.ContentHandler. These objects should be made available by the OSGi Framework to the java.net.URLConnection class.

8.2 Factories in java.net

Java provides the java.net.URL class which is used by the OSGi Framework and many of the bundles that run on the OSGi Service Platform. A key benefit of using the URL class is the ease with which a URL string is translated into a request for a resource.

The extensibility of the java.net.URL class allows new schemes (protocols) and content types to be added dynamically using java.net.URLStreamHandlerFactory objects. These new handlers allow existing applications to use new schemes and content types in the same way as the handlers provided by the Java run-time environment. This mechanism is described in the Javadoc for the URLStreamHandler and ContentHandler class, see [18] *Java .net*.

For example, the URL http://www.osgi.org/sample.txt addresses a file on the OSGi web server that is obtained with the HTTP scheme (usually a scheme provided by the Java run-time). A URL such as rsh://www.acme.com/agent.zip is addressing a ZIP file that can be obtained with the non built-in RSH scheme. A java.net.URLStreamHandlerFactory object must be registered with the java.net.URL class prior to the successful use of an RSH scheme.

There are several problems with using only the existing Java facilities for extending the handlers used by the java.net.URL class:

- *Factories Are Singleton Operations* – One java.net.URLStreamHandlerFactory object can be registered *once* with the java.net.URL class. Similarly, one java.net.ContentHandlerFactory object can be registered once with the java.net.URLConnection class. It is impossible to undo the registration of a factory or register a replacement factory.
- *Caching Of Schemes* – When a previously unused scheme is first used by the java.net.URL class, the java.net.URL class requests a java.net.URLStreamHandler object for that specific scheme from the currently registered java.net.URLStreamHandlerFactory object. A returned java.net.URLStreamHandler object is cached and subsequent requests for that scheme use the same java.net.URLStreamHandler object. This means that once a handler has been constructed for a specific scheme, this handler can no longer be removed, nor replaced, by a new handler for that scheme. This caching is likewise done for java.net.ContentHandler objects.

Both problems impact the OSGi operating model, which allows a bundle to go through different life-cycle stages that involve exposing services, removing services, updating code, replacing services provided by one bundle with services from another, etc. The existing Java mechanisms are not compatible when used by bundles.

8.3 Framework Procedures

The OSGi Framework must register a java.net.URLStreamHandlerFactory object and a java.net.ContentHandlerFactory object with the java.net.URL.setURLStreamHandlerFactory and java.net.URLConnection.setContentHandlerFactory methods, respectively.

When these two factories are registered, the OSGi Framework service registry must be tracked for the registration of URLStreamHandlerService services and java.net.ContentHandler services.

A URL Stream Handler Service must be associated with a service registration property named URL_HANDLER_PROTOCOL. The value of this url.handler.protocol property must be an array of scheme names (String[]).

A Content Handler service must be associated with a service registration property named URL_CONTENT_MIMETYPE. The value of the URL_CONTENT_MIMETYPE property must be an array of MIME types names (String[]) in the form type/subtype. See [20] *MIME Multipurpose Internet Mail Extension*.

8.3.1 Constructing a Proxy and Handler

When a URL is used with a previously unused scheme, it must query the registered java.net.URLStreamHandlerFactory object (that should have been registered by the OSGi Framework). The OSGi Framework must then search the service registry for services that are registered under URLStreamHandlerService and that match the requested scheme.

If one or more service objects are found, a proxy object must be constructed. A proxy object is necessary because the service object that provides the implementation of the java.net.URLStreamHandler object can become unregistered and Java does not provide a mechanism to withdraw a java.net.URLStreamHandler object once it is returned from a java.net.URLStreamHandlerFactory object.

Once the proxy is created, it must track the service registry for registrations and unregistrations of services matching its associated scheme. The proxy must be associated with the service that matches the scheme and has the highest value for the org.osgi.framework.Constants.SERVICE_RANKING service registration property (see *Service Registration Properties* on page 68) at any moment in time. If a proxy is associated with a URL Stream Handler Service, it must change the associated handler to a newly registered service when that service has a higher value for the ranking property.

The proxy object must forward all method requests to the associated URL Stream Handler Service until this service object becomes unregistered.

Once a proxy is created, it cannot be withdrawn because it is cached by the Java run-time. However, service objects can be withdrawn and it is possible for a proxy to exist without an associated URLStreamHandlerService/ java.net.ContentHandler object.

In this case, the proxy must handle subsequent requests until another appropriate service is registered. When this happens, the proxy class must throw a java.net.MalformedURLException exception if the signature of a method allows throwing this exception. Otherwise, a java.lang.IllegalStateException exception is thrown. This is true for both Content Handler services and URL Stream Handler Services.

Bundles must ensure that their URLStreamHandlerService or java.net.ContentHandler service objects throw these exceptions also when they have become unregistered.

Proxies for Content Handler services operate slightly differently from URL Stream Handler Service proxies. In the case that null is returned from the registered ContentHandlerFactory object, the factory will not get another chance to provide a ContentHandler object for that content-type. Thus, if there is no built-in handler, nor a registered handler for this content-type, a ContentHandler proxy must be constructed that returns the InputStream object from the URLConnection object as the content object until a handler is registered.

8.3.2 Built-in Handlers

Implementations of Java provide a number of sub-classes of java.net.URLStreamHandler classes that can handle protocols like HTTP, FTP, NEWS etc. Most Java implementations provide a mechanism to add new handlers that can be found on the classpath through class name construction.

If a registered java.net.URLStreamHandlerFactory object returns null for a built-in handler (or one that is available through the class name construction mechanism), it will never be called again for that specific scheme because the Java implementation will use its built-in handler or uses the class name construction.

It is thus not guaranteed that a registered URLStreamHandlerService object is used. Therefore, built-in handlers should take priority over handlers from the service registry to guarantee consistency. The built-in handlers, as defined in the OSGi Execution Environments (see *OSGi Defined Execution Environments* on page 428), must never be overridden.

The Content Handler Factory is implemented using a similar technique and has therefore the same problems.

To facilitate the discovery of built-in handlers that are available through the name construction, the method described in the next section must be used by the Framework before any handlers are searched for in the service registry.

8.3.3 Finding Built-in Handlers

If the system properties java.protocol.handler.pkgs or java.content.handler.pkgs are defined, they must be used to locate built-in handlers. Each property must be defined as a list of package names that are separated by a vertical bar ('|', \u007C) and that are searched in the left-to-right order (the names must *not* end in a period). For example:

```
org.osgi.impl.handlers | com.acme.url
```

The package names are the prefixes that are put in front of a scheme or content type to form a class name that can handle the scheme or content-type.

A URL Stream Handler name for a scheme is formed by appending the string ".Handler" to the scheme name. Using the packages in the previous example, the rsh scheme handler class is searched by the following names:

```
org.osgi.impl.handlers.rsh.Handler
com.acme.url.rsh.Handler
```

MIME type names contain the '/' character and can contain other characters that must not be part of a Java class name. A MIME type name must be processed as follows before it can be converted to a class name:

1. First, all slashes in the MIME name must be converted to a period ('.' \u002E). All other characters that are not allowed in a Java class name must be converted to an underscore ('_' or \u005F).

application/zip	application.zip
text/uri-list	text.uri_list
image/vnd.dwg	image.vnd_dwg

2. After this conversion, the name is appended to the list of packages specified in java.content.handler.pkgs. For example, if the content type is application/zip, and the packages are defined as in the previous example, then the following classes are searched:

   ```
   org.osgi.impl.handlers.application.zip
   com.acme.url.application.zip
   ```

The Java run-time specific packages should be listed in the appropriate properties so that implementations of the URL Stream Handler Factory and Content Handler Factory can be made aware of these packages.

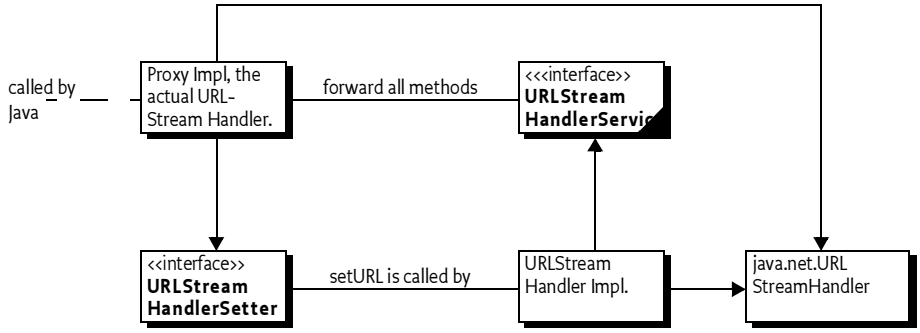
8.3.4 Protected Methods and Proxy

Implementations of java.net.URLStreamHandler class cannot be registered in the service registry for use by the proxy because the methods of the URLStreamHandler class are protected and thus not available to the proxy implementation. Also, the URLStreamHandler class checks that only the URLStreamHandler object that was returned from the URLStreamHandlerFactory object can invoke the setURL method. This means that URLStreamHandler objects in the service registry would be unable to invoke the setURL method. Invoking this method is necessary when implementing the parseURL method.

Therefore, the URLStreamHandlerService and URLStreamHandlerSetter interfaces were created. The URLStreamHandlerService interface provides public versions of the URLStreamHandler methods, except that the setURL method is missing and the parseURL method has a new first argument of type URLStreamHandlerSetter. In general, sub-classes of the URLStreamHandler class can be converted to URLStreamHandlerService classes with minimal code changes. Apart from making the relevant meth-

ods public, the parseURL method needs to be changed to invoke the setURL method on the URLStreamHandlerSetter object that the URLStreamHandlerService object was passed, rather then the setURL method of URLStreamHandler class.

Figure 25 *Proxy Issues*



To aid in the conversion of URLStreamHandler implementation classes, the AbstractURLStreamHandlerService has been provided. Apart from making the relevant methods public, the AbstractURLStreamHandlerService stores the URLStreamHandlerSetter object in a private variable. To make the setURL method work properly, it overrides the setURL method to invoke the setURL method on the saved URLStreamHandlerSetter object rather then the URLStreamHandler.setURL method. This means that a subclass of URLStreamHandler should be changed to become a sub-class of the AbstractURLStreamHandlerService class and be recompiled.

Normally, the parseURL method will have the following form:

```
class URLStreamHandlerImpl {
   ...
   protected URLStreamHandlerSetter realHandler;
   ...
   public void parseURL(
      URLStreamHandlerSetter realHandler,
         URL u, String spec, int start, int limit) {
         this.realHandler = realHandler;
         parseURL(u, spec, start, limit);
   }
   protected void setURL(URL u,
      String protocol, String host,
      int port, String authority,
      String userInfo, String path,
      String query,String ref) {
         realHandler.setURL(u, protocol, host,
            port, authority, userInfo, path,
            query, ref);
   }
   ...
}
```

The URLStreamHandler.parseURL method will call the setURL method which must be invoked on the proxy rather than this. That is why the setURL method is overridden to delegate to the URLStreamHandlerSetter object in realHandler as opposed to super.

8.4 Providing a New Scheme

The following example provides a scheme that returns the path part of the URL. The first class that is implemented is the URLStreamHandlerService. When it is started, it registers itself with the OSGi Framework. The OSGi Framework calls the openConnection method when a new java.net.URLConnection must be created. In this example, a DataConnection object is returned.

```
public class DataProtocol
   extends AbstractURLStreamHandlerService
   implements BundleActivator {
   public void start( BundleContext context ) {
      Hashtable  properties = new Hashtable();
      properties.put( URLConstants.URL_HANDLER_PROTOCOL,
         new String[] { "data" } );
      context.registerService(
         URLStreamHandlerService.class.getName(),
         this, properties );
   }
   public void stop( BundleContext context ) {}

   public URLConnection openConnection( URL url ) {
      return new DataConnection(url);
   }
}
```

The following example DataConnection class extends java.net.URLConnection and overrides the constructor so that it can provide the URL object to the super class, the connect method, and the getInputStream method. This last method returns the path part of the URL as an java.io.InputStream object.

```
class DataConnection extends java.net.URLConnection {
   DataConnection( URL url ) {super(url);}
   public void connect() {}

   public InputStream getInputStream() throws IOException {
      String s = getURL().getPath();
      byte [] buf = s.getBytes();
      return new ByteArrayInputStream(buf,1,buf.length-1);
   }
   public String getContentType() {
      return "text/plain";
   }
}
```

8.5 Providing a Content Handler

A Content Handler should extend the java.net.ContentHandler class and implement the getContent method. This method must get the InputStream object from the java.net.URLConnection parameter object and convert the bytes from this stream to the applicable type. In this example, the MIME type is text/plain and the return object is a String object.

```
public class TextPlainHandler extends ContentHandler
   implements BundleActivator {

   public void start( BundleContext context ) {
      Hashtableproperties = new Hashtable();
      properties.put( URLConstants.URL_CONTENT_MIMETYPE,
         new String[] { "text/plain" } );
      context.registerService(
         ContentHandler.class.getName(),
         this, properties );
   }
   public void stop( BundleContext context ) {}

   public Object getContent( URLConnection conn )
         throws IOException {
      InputStream in = conn.getInputStream();
      InputStreamReader r = new InputStreamReader( in );
      StringBuffer sb = new StringBuffer();
      int c;
      while ( (c=r.read()) >= o )
         sb.append( (char) c );
      r.close(); in.close();
      return sb.toString();
   }
}
```

8.6 Security Considerations

The ability to specify a protocol and add content handlers makes it possible to directly affect the behavior of a core Java VM class. The java.net.URL class is widely used by network applications and can be used by the OSGi Framework itself.

Therefore, care must be taken when providing the ability to register handlers. The two types of supported handlers are URLStreamHandlerService and java.net.ContentHandler. Only trusted bundles should be allowed to register these services and have ServicePermission[REGISTER, URLStreamHandlerService|ContentHandler] for these classes. Since these services are made available to other bundles through the java.net.URL class and java.net.URLConnection class, it is advisable to deny the use of these services (ServicePermission[GET,<name>]) to all, so that only the Framework can get them . This prevents the circumvention of the permission checks done by the java.net.URL class by using the URLStreamHandlerServices service objects directly.

8.7 org.osgi.service.url

The OSGi URL Stream and Content Handlers API 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.url; specification-version=1.0
```

8.7.1 Summary

- AbstractURLStreamHandlerService – Abstract implementation of the URLStreamHandlerService interface. [p.165]
- URLConstants – Defines standard names for property keys associated with URLStreamHandlerService[p.167] and java.net.ContentHandler services. [p.166]
- URLStreamHandlerService – Service interface with public versions of the protected java.net.URLStreamHandler methods. [p.167]
- URLStreamHandlerSetter – Interface used by URLStreamHandlerService objects to call the setURL method on the proxy URLStreamHandler object. [p.168]

8.7.2 public abstract class AbstractURLStreamHandlerService extends URLStreamHandler implements URLStreamHandlerService

Abstract implementation of the URLStreamHandlerService interface. All the methods simply invoke the corresponding methods on java.net.URLStreamHandler except for parseURL and setURL, which use the URLStreamHandlerSetter parameter. Subclasses of this abstract class should not need to override the setURL and parseURL(URLStreamHandlerSetter,...) methods.

8.7.2.1 protected URLStreamHandlerSetter realHandler

The URLStreamHandlerSetter object passed to the parseURL method.

8.7.2.2 public AbstractURLStreamHandlerService( )

8.7.2.3 public boolean equals(URL u1, URL u2)

- This method calls super.equals(URL,URL).

See Also java.net.URLStreamHandler.equals(URL,URL)

8.7.2.4 public int getDefaultPort( )

- This method calls super.getDefaultPort.

See Also java.net.URLStreamHandler.getDefaultPort

8.7.2.5 public InetAddress getHostAddress(URL u)

- This method calls super.getHostAddress.

See Also java.net.URLStreamHandler.getHostAddress

8.7.2.6 public int hashCode(URL u)

- This method calls `super.hashCode(URL)`.

See Also `java.net.URLStreamHandler.hashCode(URL)`

8.7.2.7 public boolean hostsEqual(URL u1, URL u2)

- This method calls `super.hostsEqual`.

See Also `java.net.URLStreamHandler.hostsEqual`

8.7.2.8 public abstract URLConnection openConnection(URL u) throws IOException

See Also `java.net.URLStreamHandler.openConnection`

8.7.2.9 public void parseURL(URLStreamHandlerSetter realHandler, URL u, String spec, int start, int limit)

realHandler The object on which the `setURL` method must be invoked for the specified URL.

- Parse a URL using the `URLStreamHandlerSetter` object. This method sets the `realHandler` field with the specified `URLStreamHandlerSetter` object and then calls `parseURL(URL, String, int, int)`.

See Also `java.net.URLStreamHandler.parseURL`

8.7.2.10 public boolean sameFile(URL u1, URL u2)

- This method calls `super.sameFile`.

See Also `java.net.URLStreamHandler.sameFile`

8.7.2.11 protected void setURL(URL u, String proto, String host, int port, String file, String ref)

- This method calls `realHandler.setURL(URL, String, String, int, String, String)`.

See Also `java.net.URLStreamHandler.setURL(URL, String, String, int, String, String)`

Deprecated This method is only for compatibility with handlers written for JDK 1.1.

8.7.2.12 protected void setURL(URL u, String proto, String host, int port, String auth, String user, String path, String query, String ref)

- This method calls `realHandler.setURL(URL, String, String, int, String, String, String, String)`.

See Also `java.net.URLStreamHandler.setURL(URL, String, String, int, String, String, String, String)`

8.7.2.13 public String toExternalForm(URL u)

- This method calls `super.toExternalForm`.

See Also `java.net.URLStreamHandler.toExternalForm`

8.7.3 public interface URLConstants

Defines standard names for property keys associated with URLStreamHandlerService[p.167] and `java.net.ContentHandler` services.

The values associated with these keys are of type java.lang.String[], unless otherwise indicated.

8.7.3.1 public static final String URL_CONTENT_MIMETYPE = "url.content.mimetype"

Service property naming the MIME types serviced by a java.net.ContentHandler. The property's value is an array of MIME types.

8.7.3.2 public static final String URL_HANDLER_PROTOCOL = "url.handler.protocol"

Service property naming the protocols serviced by a URLStreamHandlerService. The property's value is an array of protocol names.

8.7.4 public interface URLStreamHandlerService

Service interface with public versions of the protected java.net.URLStreamHandler methods.

The important differences between this interface and the URLStreamHandler class are that the setURL method is absent and the parseURL method takes a URLStreamHandlerSetter[p.168] object as the first argument. Classes implementing this interface must call the setURL method on the URLStreamHandlerSetter object received in the parseURL method instead of URLStreamHandler.setURL to avoid a SecurityException.

See Also AbstractURLStreamHandlerService[p.165]

8.7.4.1 public boolean equals(URL u1, URL u2)

See Also java.net.URLStreamHandler.equals(URL, URL)

8.7.4.2 public int getDefaultPort( )

See Also java.net.URLStreamHandler.getDefaultPort

8.7.4.3 public InetAddress getHostAddress(URL u)

See Also java.net.URLStreamHandler.getHostAddress

8.7.4.4 public int hashCode(URL u)

See Also java.net.URLStreamHandler.hashCode(URL)

8.7.4.5 public boolean hostsEqual(URL u1, URL u2)

See Also java.net.URLStreamHandler.hostsEqual

8.7.4.6 public URLConnection openConnection(URL u) throws IOException

See Also java.net.URLStreamHandler.openConnection

8.7.4.7 public void parseURL(URLStreamHandlerSetter realHandler, URL u, String spec, int start, int limit)

realHandler The object on which setURL must be invoked for this URL.

- Parse a URL. This method is called by the URLStreamHandler proxy, instead of java.net.URLStreamHandler.parseURL, passing a URLStreamHandlerSetter object.

See Also java.net.URLStreamHandler.parseURL

8.7.4.8 public boolean sameFile(URL u1, URL u2)

See Also java.net.URLStreamHandler.sameFile

8.7.4.9 public String toExternalForm(URL u)

See Also java.net.URLStreamHandler.toExternalForm

8.7.5 public interface URLStreamHandlerSetter

Interface used by URLStreamHandlerService objects to call the setURL method on the proxy URLStreamHandler object.

Objects of this type are passed to the URLStreamHandlerService.parseURL[p.167] method. Invoking the setURL method on the URLStreamHandlerSetter object will invoke the setURL method on the proxy URLStreamHandler object that is actually registered with java.net.URL for the protocol.

8.7.5.1 public void setURL(URL u, String protocol, String host, int port, String file, String ref)

See Also java.net.URLStreamHandler.setURL(URL, String, String, int, String, String)

Deprecated This method is only for compatibility with handlers written for JDK 1.1.

8.7.5.2 public void setURL(URL u, String protocol, String host, int port, String authority, String userInfo, String path, String query, String ref)

See Also java.net.URLStreamHandler.setURL(URL, String, String, int, String, String, String, String)

8.8 References

[18] *Java .net*
http://java.sun.com/j2se/1.4/docs/api/java/net/package-summary.html

[19] *URLs*
http://www.ietf.org/rfc/rfc1738.txt

[20] *MIME Multipurpose Internet Mail Extension*
http://www.nacs.uci.edu/indiv/ehood/MIME/MIME.html

[21] *Assigned MIME Media Types*
http://www.iana.org/assignments/media-types

9 Log Service Specification

Version 1.2

9.1 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.

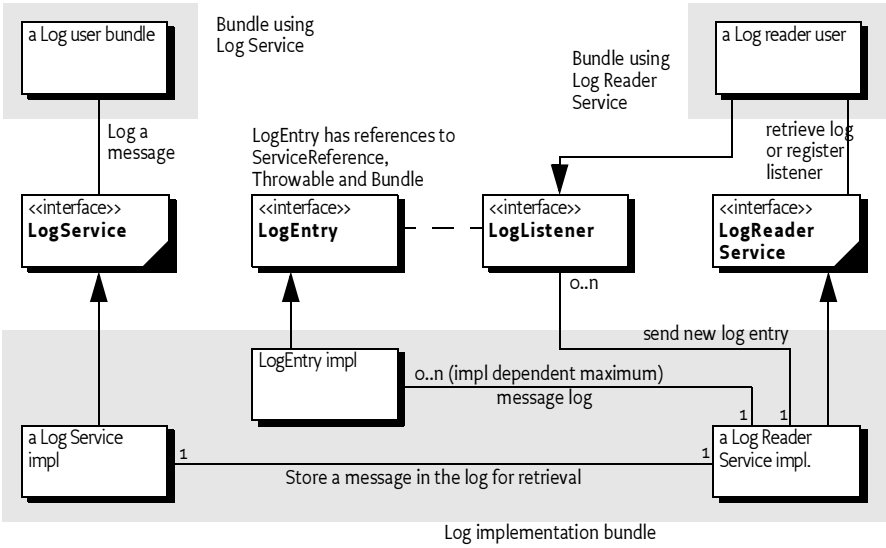
This specification defines the methods and semantics of interfaces which bundle developers can use to log entries and to retrieve log entries.

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.

9.1.1 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 26 Log Service Class Diagram org.osgi.service.log package



9.2 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.

9.3 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 6 *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 6 *Log Levels*

9.4 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.

9.5 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.

9.6 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.

9.6.1 Bundle Events Mapping

A Bundle Event is mapped to a LogEntry object according to Table 7, “Mapping of Bundle Events to Log Entries,” on page 173.

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 7 *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 7 *Mapping of Bundle Events to Log Entries*

9.6.2 Service Events Mapping

A Service Event is mapped to a LogEntry object according to Table 8, "Mapping of Service Events to Log Entries," on page 174.

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 8 *Mapping of Service Events to Log Entries*

9.6.3 Framework Events Mapping

A Framework Event is mapped to a LogEntry object according to Table 9, "Mapping of Framework Event to Log Entries," on page 175.

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 9 *Mapping of Framework Event to Log Entries*

9.7 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.

9.8 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.

9.9 org.osgi.service.log

The OSGi Log Service Package. Specification Version 1.2.

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

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

9.9.1 Summary

- *LogEntry* – Provides methods to access the information contained in an individual Log Service log entry. [p.169]
- *LogListener* – Subscribes to `LogEntry` objects from the `LogReaderService`. [p.169]
- *LogReaderService* – Provides methods to retrieve `LogEntry` objects from the log. [p.169]
- *LogService* – Provides methods for bundles to write messages to the log. [p.169]

9.9.2 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.178] , `LogListener`[p.169]

9.9.2.1 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.

9.9.2.2 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` objcet.

9.9.2.3 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.179], LogService.LOG_WARNING[p.179], LogService.LOG_INFO[p.179], LogService.LOG_DEBUG[p.178]

9.9.2.4 public String getMessage( )

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

Returns String containing the message associated with this LogEntry object.

9.9.2.5 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.

9.9.2.6 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()

9.9.3 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.169], LogEntry[p.169], LogReaderService.addLogListener(LogListener)[p.178], LogReaderService.removeLogListener(LogListener)[p.178]

9.9.3.1 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.169]

9.9.4 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.169], LogListener[p.169], LogListener.logged(LogEntry)[p.177]

9.9.4.1 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.169], LogEntry[p.169], LogListener.logged(LogEntry)[p.177]

9.9.4.2 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.

9.9.4.3 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.169]

9.9.5 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.179]
2 LOG_WARNING[p.179]
3 LOG_INFO[p.179]
4 LOG_DEBUG[p.178]

9.9.5.1 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.

9.9.5.2 public static final int LOG_ERROR = 1

An error message (Value 1).

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

9.9.5.3 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.

9.9.5.4 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.

9.9.5.5 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.179], `LOG_WARNING`[p.179], `LOG_INFO`[p.179], `LOG_DEBUG`[p.178]

9.9.5.6 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.179], `LOG_WARNING`[p.179], `LOG_INFO`[p.179], `LOG_DEBUG`[p.178]

9.9.5.7 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.179], LOG_WARNING[p.179], LOG_INFO[p.179], LOG_DEBUG[p.178]

9.9.5.8 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.179], LOG_WARNING[p.179], LOG_INFO[p.179], LOG_DEBUG[p.178]

10 Configuration Admin Service Specification

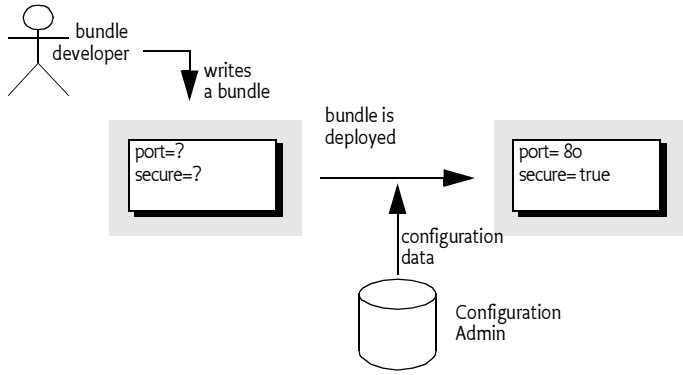
Version 1.1

10.1 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 27 *Configuration Admin Service Overview*



10.1.1 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.

10.1.2 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.

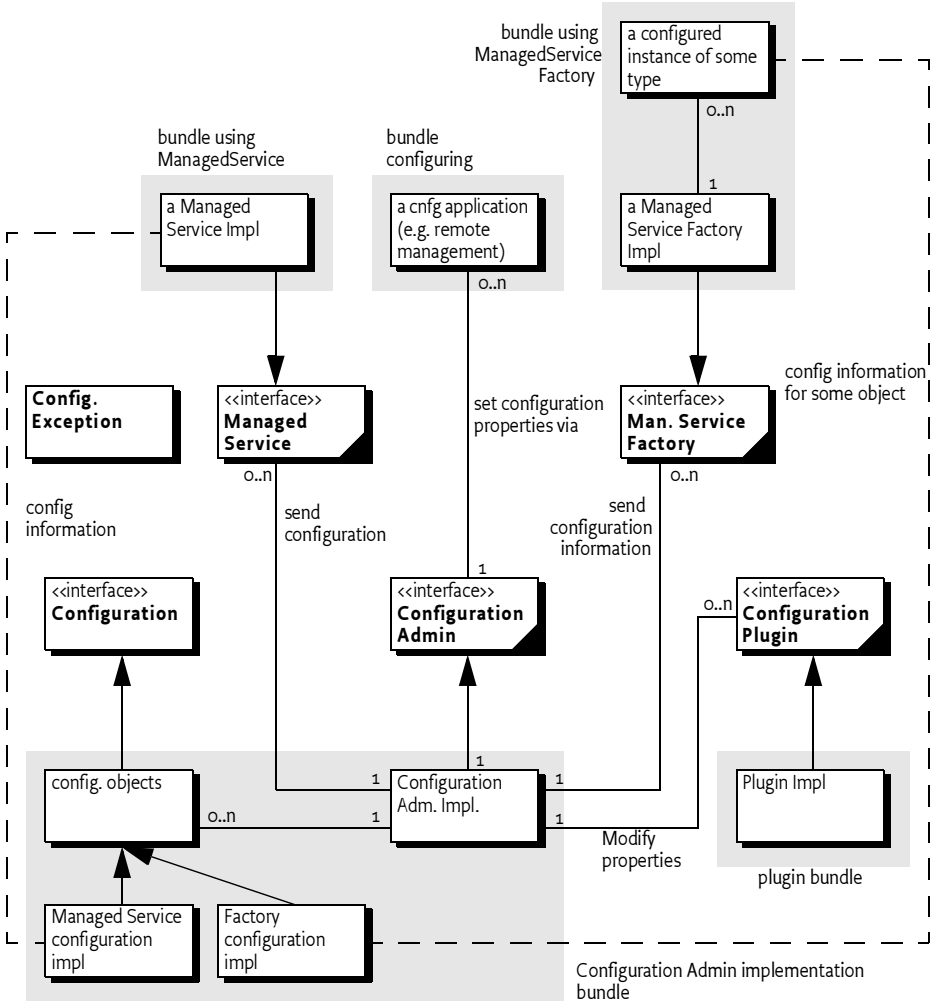
The database can be manipulated either by the Management Agent or bundles that configure themselves.

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.

10.1.3 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 28 Configuration Admin Class Diagram org.osgi.service.cm



10.2 Configuration Targets

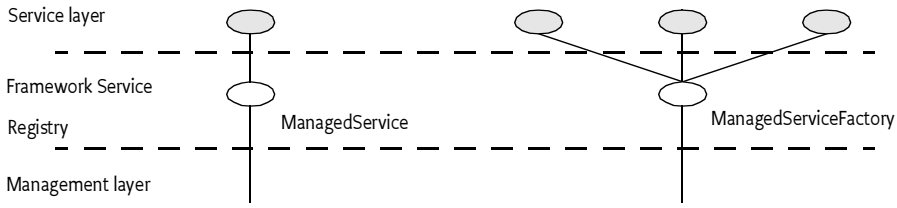
One of the more complicated aspects of this specification is the subtle distinction between the ManagedService and ManagedServiceFactory classes.

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 29 *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.

10.3 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.

10.3.1 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.

10.3.1.1 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.

10.3.1.2 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.

10.3.1.3 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 10 Schemes for Device-Oriented PID Names

10.4 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 185 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.

10.4.1 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.

10.4.2 Configuration Properties

A configuration dictionary contains a set of properties in a Dictionary object. The value types that must be used are the same types as the types supported in the Framework service registry, which are defined as:

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

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

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

vector = <Vector of type>
```

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

Bundles should not use nested vectors or arrays, nor should they use mixed types. Using mixed types or nesting makes it impossible to use the meta typing specification. See *Metatype Specification* on page 377.

10.4.3 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.

10.4.4 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 201, 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.

10.4.5 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.

10.5 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.

10.5.1 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.

10.5.2 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.

10.5.3 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 204.

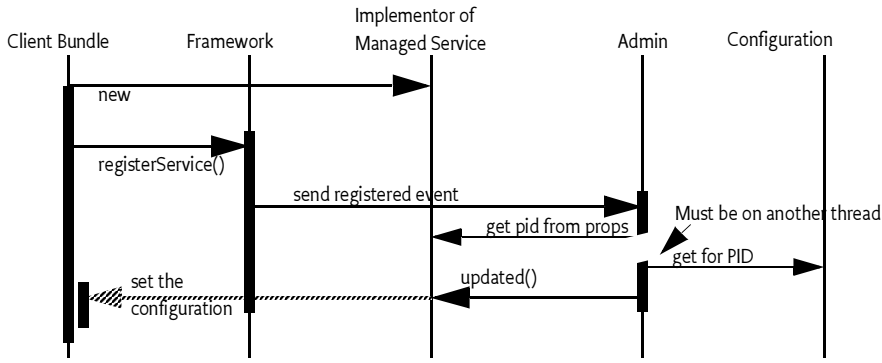
When this registration is detected by the Configuration Admin service, the following steps must occur:

- 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 30 *Managed Service Configuration Action Diagram*



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

10.5.4 Race Conditions

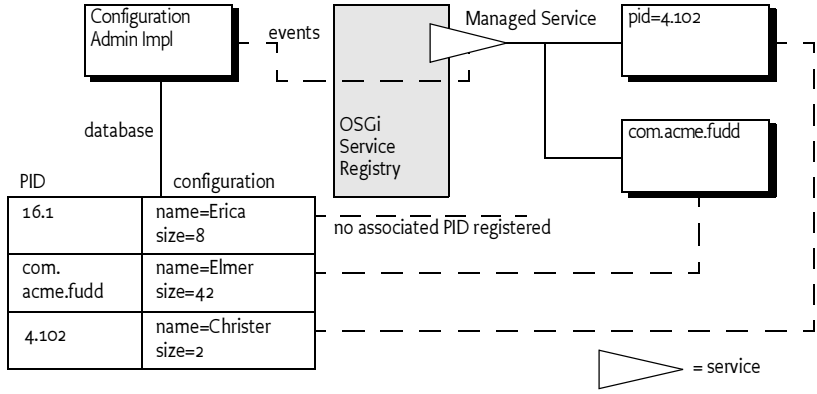
When a Managed Service is registered, the default properties may be visible in the service registry for a short period before they are replaced by the properties of the actual configuration dictionary. Care should be taken that this visibility does not cause race conditions for other bundles.

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.

10.5.5 Examples of Managed Service

Figure 31 shows a Managed Service configuration example. Two services are registered under the ManagedService interface, each with a different PID.

Figure 31 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.

10.5.5.1 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
  }
```

10.5.6 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.

10.6 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.

10.6.1 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.

10.6.1.1 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).

10.6.1.2 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.

10.6.1.3 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.

10.6.2 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.

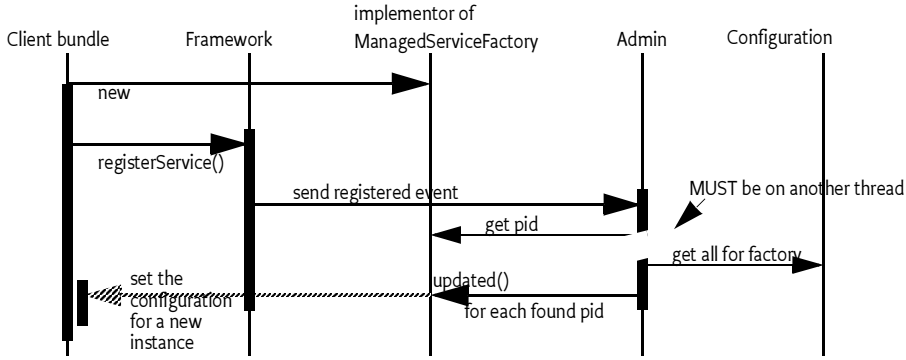
When the Configuration Admin service detects the registration of a Managed Service Factory, it must find all configuration dictionaries for this factory and must then sequentially call ManagedServiceFactory.updated(String,Dictionary) for each configuration dictionary. The first argument is the PID of 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 32 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.

10.6.3 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.

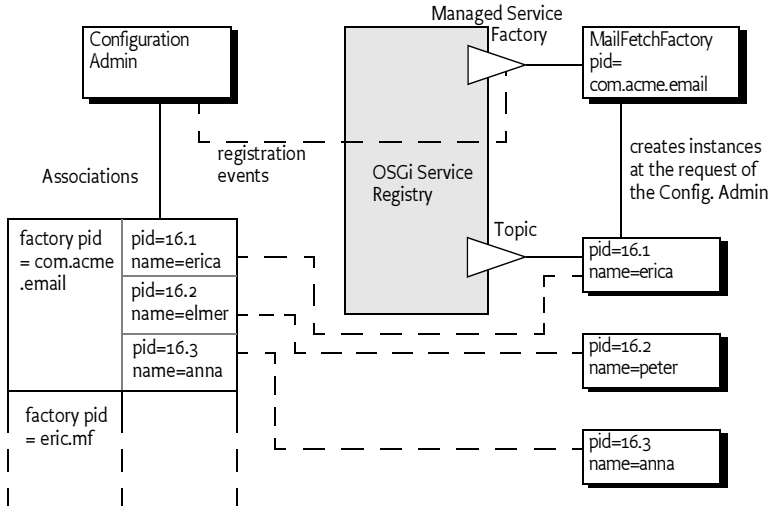
10.6.4 Managed Service Factory Example

Figure 33 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 33 *Managed Service Factory Example*



10.6.5 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();
         }
      }
   }
```

10.7 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.

10.7.1 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.

10.7.2 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.

10.7.3 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.

10.7.3.1 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.

10.7.4 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.

10.7.5 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.

10.8 Configuration Plugin

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

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

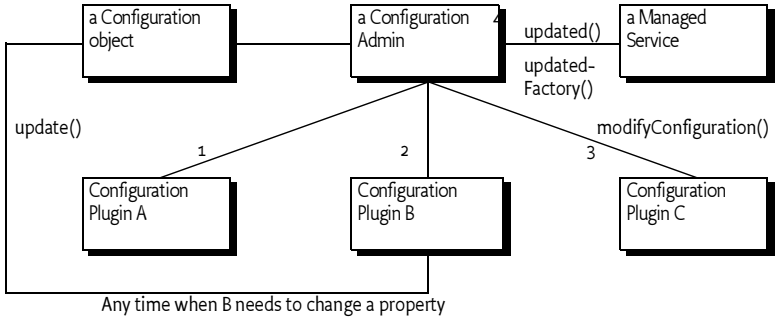
The ConfigurationPlugin interface has only one method: modifyConfiguration(ServiceReference,Dictionary). This method inspects or modifies configuration data.

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

ConfigurationPlugin objects should not modify properties that belong to the configuration properties of the target service unless the implications are understood. This functionality is mainly intended to provide functions that leverage the Framework service registry. The changes made by the plugin should normally not be validated. However, the Configuration Admin must ignore changes to the automatic properties as described in *Automatic Properties* on page 189.

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 34 *Order of Configuration Plugin Services*



10.8.1 Limiting The Targets

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

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

10.8.2 Example of Property Expansion

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

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

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

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

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

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

10.8.3 Configuration Data Modifications

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

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

10.8.4 Forcing a Callback

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

10.8.5 Calling Order

The order in which the ConfigurationPlugin objects are called must depend on the service.cmRanking configuration property of the ConfigurationPlugin object. Table 11 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 11 service.cmRanking *Usage For Ordering*

10.9 Remote Management

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

The Configuration Admin service is a primary aspect of remote management, however, and this specification must be compatible with common remote management standards. This section discusses some of the issues of using this specification with [22] *DMTF Common Information Model* (CIM) and [23] *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.

10.9.1 Common Information Model

Common Information Model (CIM) defines the managed objects in [25] *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 [24] *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 [26] *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.

10.9.2 Simple Network Management Protocol

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

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

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

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

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

10.10 Meta Typing

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

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

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

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

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

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

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

10.11 Security

10.11.1 Permissions

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

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

Table 12 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.

10.11.2 Forging PIDs

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

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

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

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

10.11.3 Configuration and Permission Administration

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

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

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

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

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

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

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

10.12 Configurable Service

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

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

This scheme has the following disadvantages:

- *No factory* – Only registered services can be configured, unlike the Managed Service Factory that configures any number of services.
- *Atomicity* – The beans or reflection API can only modify one property at a time and there is no way to tell the bean that no more modifications to the properties will follow. This limitation complicates updates of configurations that have dependencies between properties.
  This specification passes a Dictionary object that sets all the configuration properties atomically.

- *Profile* – The Java beans API is linked to many packages that are not likely to be present in OSGi environments. The reflection API may be present but is not simple to use.
  This specification has no required libraries.
- *User Interface support* – UI support in beans is very rudimentary when no AWT is present.
  The associated Metatyping specification does not require any external libraries, and has extensive support for UIs including localization.

10.13 Changes

10.13.1 Clarifications

- It was not clear from the description that a PID received through a Managed Service Factory must not be used to register a Managed Service. This has been highlighted in the appropriate sections.
- It was not clearly specified that a call-back to a target only happens when the data is updated or the target is registered. The creation of a Configuration object does not initiate a call-back. This has been highlighted in the appropriate sections.
- In this release, when a bundle is uninstalled, all Configuration objects that are dynamically bound to that bundle must be unbound again. See *Location Binding* on page 187.
- 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.

10.13.2 Removal of Bundle Location Property

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

10.13.3 Plug-in Usage

It was not completely clear when a plug-in must be called and how the properties dictionary should behave. This has been clearly specified in *Configuration Plugin* on page 201.

10.13.4 BigInteger/BigDecimal

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

10.13.5 Equals

The behavior of the equals and hashCode methods is now defined. See *Equality* on page 189.

10.13.6 Constant for service.factoryPid

Added a new constant in the ConfigurationAdmin class. See *SERVICE_FACTORYPID* on page 213. This caused this specification to step from version 1.0 to version 1.1.

10.14 org.osgi.service.cm

The OSGi Configuration Admin service 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.cm; specification-version=1.1
```

10.14.1 Summary

- *Configuration* – The configuration information for a ManagedService or ManagedServiceFactory object. [p.209]
- *ConfigurationAdmin* – Service for administering configuration data. [p.212]
- *ConfigurationException* – An Exception class to inform the Configuration Admin service of problems with configuration data. [p.215]
- *ConfigurationPlugin* – A service interface for processing configuration dictionary before the update. [p.216]
- *ManagedService* – A service that can receive configuration data from a Configuration Admin service. [p.217]
- *ManagedServiceFactory* – Manage multiple service instances. [p.219]

10.14.2 public interface Configuration

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

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

The properties handled in this configuration have case insensitive String objects as keys. However, case is preserved from the last set key/value. The value of the property may be of the following types:

```
 type        ::=
    String     | Integer    | Long
  | Float      | Double     | Byte
  | Short      | Character  | Boolean
  | vector     | arrays
 primitive   ::= long | int | short | char | byte | double |
float
```

```
arrays      ::= primitive '[]' | type '[]' | null
vector      ::= Vector of type or null
```

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

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.

10.14.2.1 public void delete( ) throws IOException

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

Throws `IOException` – If delete fails

`IllegalStateException` – if this configuration has been deleted

10.14.2.2 public boolean equals(Object other)

other `Configuration` object to compare against

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

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

10.14.2.3 public String getBundleLocation( )

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

  This call requires `AdminPermission`.

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

Throws `SecurityException` – if the caller does not have `AdminPermission`.

`IllegalStateException` – if this `Configuration` object has been deleted.

10.14.2.4 public String getFactoryPid( )

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

Returns factory PID or null

Throws IllegalStateException – if this configuration has been deleted

10.14.2.5 public String getPid( )

- Get the PID for this Configuration object.

Returns the PID for this Configuration object.

Throws IllegalStateException – if this configuration has been deleted

10.14.2.6 public Dictionary getProperties( )

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

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

Returns A private copy of the properties for the caller or null. These properties must not contain the "service.bundleLocation" property. The value of this property may be obtained from the getBundleLocation method.

Throws IllegalStateException – if this configuration has been deleted

10.14.2.7 public int hashCode( )

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

Returns hash code for this Configuration object

10.14.2.8 public void setBundleLocation(String bundleLocation)

bundleLocation a bundle location or null

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

  This method requires AdminPermission.

Throws SecurityException – if the caller does not have AdminPermission

IllegalStateException – if this configuration has been deleted

10.14.2.9 public void update(Dictionary properties) throws IOException

properties the new set of properties for this configuration

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

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

  These system properties are all of type String.

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

Throws `IOException` – if update cannot be made persistent

`IllegalArgumentException` – if the `Dictionary` object contains invalid configuration types

`IllegalStateException` – if this configuration has been deleted

10.14.2.10 public void update( ) throws IOException

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

  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.216]

10.14.3 public interface ConfigurationAdmin

Service for administering configuration data.

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

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

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

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

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

When the ConfigurationAdmin detects the registration of a Managed Service, it checks its persistent storage for a configuration object whose PID matches the PID registration property (`service.pid`) of the Managed Service. If found, it calls `ManagedService.updated`[p.219] 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.

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

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

Since 1.1

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

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

Since 1.1

10.14.3.3 public Configuration createFactoryConfiguration(String factoryPid)

throws IOException

factoryPid PID of factory (not null).

- Create a new factory Configuration object with a new PID. The properties of the new Configuration object are null until the first time that its Configuration.update[p.211] 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.

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

factoryPid PID of factory (not null).

location a bundle location string, or null.

- Create a new factory Configuration object with a new PID. The properties of the new Configuration object are null until the first time that its Configuration.update[p.211] 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.

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

pid persistent identifier.

location the bundle location string, or null.

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

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

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

  This method requires AdminPermission.

Returns an existing or new Configuration object.

Throws IOException – if access to persistent storage fails.

SecurityException – if the caller does not have AdminPermission.

10.14.3.6 public Configuration getConfiguration(String pid) throws IOException

pid persistent identifier.

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

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

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

Returns an existing or new Configuration matching the PID.

Throws IOException – if access to persistent storage fails.

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

10.14.3.7 public Configuration[] listConfigurations(String filter) throws IOException, InvalidSyntaxException

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

- List the current Configuration objects which match the filter.

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

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

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

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

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

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

Throws IOException – if access to persistent storage fails

InvalidSyntaxException – if the filter string is invalid

10.14.4 public class ConfigurationException extends Exception

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

10.14.4.1 public ConfigurationException(String property, String reason)

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

reason reason for failure

- Create a ConfigurationException object.

10.14.4.2 public String getProperty( )

- Return the property name that caused the failure or null.

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

10.14.4.3 public String getReason( )

- Return the reason for this exception.

Returns reason of the failure

10.14.5 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.

10.14.5.1 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.

10.14.5.2 public void modifyConfiguration(ServiceReference reference, Dictionary properties)

reference reference to the Managed Service or Managed Service Factory

configuration 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.

10.14.6 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.

10.14.6.1 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

10.14.7 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 );
  }
  ...
}
```

10.14.7.1 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.

10.14.7.2 public String getName( )

- Return a descriptive name of this factory.

Returns the name for the factory, which might be localized

10.14.7.3 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.215] 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.

10.15 References

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

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

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

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

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

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

11 Device Access Specification

Version 1.1

11.1 Introduction

A Service Platform is a meeting point for services and devices from many different vendors: a meeting point where users add and cancel service subscriptions, newly installed services find their corresponding input and output devices, and device drivers connect to their hardware.

In an OSGi Service Platform, these activities will dynamically take place while the Framework is running. Technologies such as USB and IEEE 1394 explicitly support plugging and unplugging devices at any time, and wireless technologies are even more dynamic.

This flexibility makes it hard to configure all aspects of an OSGi Service Platform, particularly those relating to devices. When all of the possible services and device requirements are factored in, each OSGi Service Platform will be unique. Therefore, automated mechanisms are needed that can be extended and customized, in order to minimize the configuration needs of the OSGi environment.

The Device Access specification supports the coordination of automatic detection and attachment of existing devices on an OSGi Service Platform, facilitates hot-plugging and -unplugging of new devices, and downloads and installs device drivers on demand.

This specification, however, deliberately does not prescribe any particular device or network technology, and mentioned technologies are used as examples only. Nor does it specify a particular device discovery method. Rather, this specification focuses on the attachment of devices supplied by different vendors. It emphasizes the development of standardized device interfaces to be defined in device categories, although no such device categories are defined in this specification.

11.1.1 Essentials

- *Embedded Devices* – OSGi bundles will likely run in embedded devices. This environment implies limited possibility for user interaction, and low-end devices will probably have resource limitations.
- *Remote Administration* – OSGi environments must support administration by a remote service provider.
- *Vendor Neutrality* – OSGi-compliant driver bundles will be supplied by different vendors; each driver bundle must be well-defined, documented, and replaceable.

- *Continuous Operation* – OSGi environments will be running for extended periods without being restarted, possibly continuously, requiring stable operation and stable resource consumption.
- *Dynamic Updates* – As much as possible, driver bundles must be individually replaceable without affecting unrelated bundles. In particular, the process of updating a bundle should not require a restart of the whole OSGi Service Platform or disrupt operation of connected devices.

A number of requirements must be satisfied by Device Access implementations in order for them to be OSGi-compliant. Implementations must support the following capabilities:

- *Hot-Plugging* – Plugging and unplugging of devices at any time if the underlying hardware and drivers allow it.
- *Legacy Systems* – Device technologies which do not implement the automatic detection of plugged and unplugged devices.
- *Dynamic Device Driver Loading* – Loading new driver bundles on demand with no prior device-specific knowledge of the Device service.
- *Multiple Device Representations* – Devices to be accessed from multiple levels of abstraction.
- *Deep Trees* – Connections of devices in a tree of mixed network technologies of arbitrary depth.
- *Topology Independence* – Separation of the interfaces of a device from where and how it is attached.
- *Complex Devices* – Multifunction devices and devices that have multiple configurations.

11.1.2 Operation

This specification defines the behavior of a device manager (which is *not* a service as might be expected). This device manager detects registration of Device services and is responsible for associating these devices with an appropriate Driver service. These tasks are done with the help of Driver Locator services and the Driver Selector service that allow a device manager to find a Driver bundle and install it.

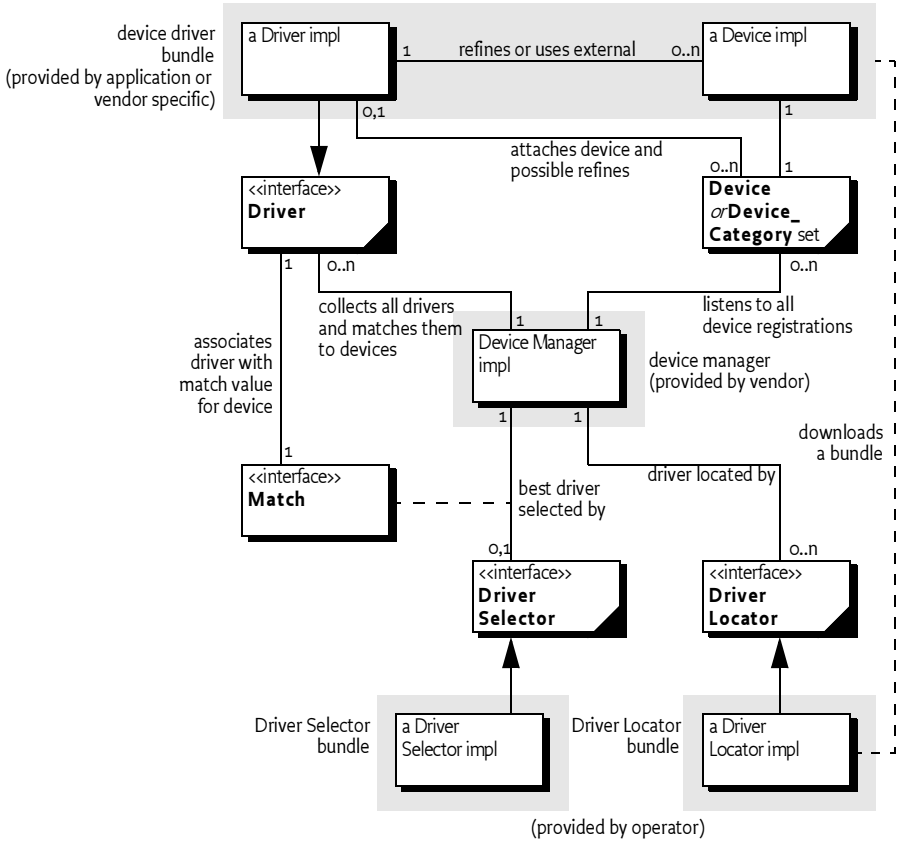
11.1.3 Entities

The main entities of the Device Access specification are:

- *Device Manager* – The bundle that controls the initiation of the attachment process behind the scenes.
- *Device Category* – Defines how a Driver service and a Device service can cooperate.
- *Driver* – Competes for attaching Device services of its recognized device category. See *Driver Services* on page 230.
- *Device* – A representation of a physical device or other entity that can be attached by a Driver service. See *Device Services* on page 225.
- *DriverLocator* – Assists in locating bundles that provide a Driver service. See *Driver Locator Service* on page 237.
- *DriverSelector* – Assists in selecting which Driver service is best suited to a Device service. See *The Driver Selector Service* on page 239.

Figure 35 show the classes and their relationships.

Figure 35 *Device Access Class Overview*



11.2 Device Services

A Device service represents some form of a device. It can represent a hardware device, but that is not a requirement. Device services differ widely: some represent individual physical devices and others represent complete networks. Several Device services can even simultaneously represent the same physical device at different levels of abstraction. For example:

- A USB network.
- A device attached on the USB network.
- The same device recognized as a USB to Ethernet bridge.
- A device discovered on the Ethernet using Salutation.
- The same device recognized as a simple printer.
- The same printer refined to a PostScript printer.

A device can also be represented in different ways. For example, a USB mouse can be considered as:

- A USB device which delivers information over the USB bus.
- A mouse device which delivers x and y coordinates and information about the state of its buttons.

Each representation has specific implications:

- That a particular device is a mouse is irrelevant to an application which provides management of USB devices.
- That a mouse is attached to a USB bus or a serial port would be inconsequential to applications that respond to mouse-like input.

Device services must belong to a defined *device category*, or else they can implement a generic service which models a particular device, independent of its underlying technology. Examples of this type of implementation could be Sensor or Actuator services.

A device category specifies the methods for communicating with a Device service, and enables interoperability between bundles that are based on the same underlying technology. Generic Device services will allow interoperability between bundles that are not coupled to specific device technologies.

For example, a device category is required for the USB, so that Driver bundles can be written that communicate to the devices that are attached to the USB. If a printer is attached, it should also be available as a generic Printer service defined in a Printer service specification, indistinguishable from a Printer service attached to a parallel port. Generic categories, such as a Printer service, should also be described in a Device Category.

It is expected that most Device service objects will actually represent a physical device in some form, but that is not a requirement of this specification. A Device service is represented as a normal service in the OSGi Framework and all coordination and activities are performed upon Framework services. This specification does not limit a bundle developer from using Framework mechanisms for services that are not related to physical devices.

11.2.1 Device Service Registration

A Device service is defined as a normal service registered with the Framework that either:

- Registers a service object under the interface org.osgi.service.Device with the Framework, or
- Sets the DEVICE_CATEGORY property in the registration. The value of DEVICE_CATEGORY is an array of String objects of all the device categories that the device belongs to. These strings are defined in the associated device category.

If this document mentions a Device service, it is meant to refer to services registered with the name org.osgi.service.device.Device *or* services registered with the DEVICE_CATEGORY property set.

When a Device service is registered, additional properties may be set that describe the device to the device manager and potentially to the end users. The following properties have their semantics defined in this specification:

- DEVICE_CATEGORY – A marker property indicating that this service must be regarded as a Device service by the device manager. Its value is of type String[], and its meaning is defined in the associated device category specification.
- DEVICE_DESCRIPTION – Describes the device to an end user. Its value is of type String.

- DEVICE_SERIAL – A unique serial number for this device. If the device hardware contains a serial number, the driver bundle is encouraged to specify it as this property. Different Device services representing the same physical hardware at different abstraction levels should set the same DEVICE_SERIAL, thus simplifying identification. Its value is of type String.
- service.pid – Service Persistent ID (PID), defined in org.osgi.framework.Constants. Device services should set this property. It must be unique among all registered services. Even different abstraction levels of the same device must use different PIDs. The service PIDs must be reproducible, so that every time the same hardware is plugged in, the same PIDs are used.

11.2.2 Device Service Attachment

When a Device service is registered with the Framework, the device manager is responsible for finding a suitable Driver service and instructing it to attach to the newly registered Device service. The Device service itself is passive: it only registers a Device service with the Framework and then waits until it is called.

The actual communication with the underlying physical device is not defined in the Device interface because it differs significantly between different types of devices. The Driver service is responsible for attaching the device in a device type-specific manner. The rules and interfaces for this process must be defined in the appropriate device category.

If the device manager is unable to find a suitable Driver service, the Device service remains unattached. In that case, if the service object implements the Device interface, it must receive a call to the noDriverFound() method. The Device service can wait until a new driver is installed, or it can unregister and attempt to register again with different properties that describe a more generic device or try a different configuration.

11.2.2.1 Idle Device Service

The main purpose of the device manager is to try to attach drivers to idle devices. For this purpose, a Device service is considered *idle* if no bundle that itself has registered a Driver service is using the Device service.

11.2.2.2 Device Service Unregistration

When a Device service is unregistered, no immediate action is required by the device manager. The normal service of unregistering events, provided by the Framework, takes care of propagating the unregistration information to affected drivers. Drivers must take the appropriate action to release this Device service and perform any necessary cleanup, as described in their device category specification.

The device manager may, however, take a device unregistration as an indication that driver bundles may have become idle and are thus eligible for removal. It is therefore important for Device services to unregister their service object when the underlying entity becomes unavailable.

11.3 Device Category Specifications

A device category specifies the rules and interfaces needed for the communication between a Device service and a Driver service. Only Device services and Driver services of the same device category can communicate and cooperate.

The Device Access service specification is limited to the attachment of Device services by Driver services, and does *not* enumerate different device categories.

Other specifications must specify a number of device categories before this specification can be made operational. Without a set of defined device categories, no interoperability can be achieved.

Device categories are related to a specific device technology, such as USB, IEEE 1394, JINI, UPnP, Salutation, CEBus, Lonworks, and others. The purpose of a device category specification is to make all Device services of that category conform to an agreed interface, so that, for example, a USB Driver service of vendor A can control Device services from vendor B attached to a USB bus.

This specification is limited to defining the guidelines for device category definitions only. Device categories may be defined by the OSGi organization or by external specification bodies – for example, when these bodies are associated with a specific device technology.

11.3.1 Device Category Guidelines

A device category definition comprises the following elements:

- An interface that all devices belonging to this category must implement. This interface should lay out the rules of how to communicate with the underlying device. The specification body may define its own device interfaces (or classes) or leverage existing ones. For example, a serial port device category could use the javax.comm.SerialPort interface which is defined in [28] *Java Communications API*.
  When registering a device belonging to this category with the Framework, the interface or class name for this category must be included in the registration.
- A set of service registration properties, their data types, and semantics, each of which must be declared as either MANDATORY or OPTIONAL for this device category.
- A range of match values specific to this device category. Matching is explained later in *The Device Attachment Algorithm* on page 241.

11.3.2 Sample Device Category Specification

The following is a partial example of a fictitious device category:

```
public interface /* com.acme.widget.*/ WidgetDevice {
   int MATCH_SERIAL                    = 10;
   int MATCH_VERSION                   =  8;
   int MATCH_MODEL                     =  6;
   int MATCH_MAKE                      =  4;
   int MATCH_CLASS                     =  2;
```

```
    void sendPacket( byte [] data );
    byte [] receivePacket( long timeout );
  }
```

Devices in this category must implement the interface com.acme.widget.WidgetDevice to receive attachments from Driver services in this category.

Device properties for this fictitious category are defined in table Table 13.

Property name	M/O	Type	Value
DEVICE_CATEGORY	M	String[]	{"Widget"}
com.acme.class	M	String	A class description of this device. For example "audio", "video", "serial", etc. An actual device category specification should contain an exhaustive list and define a process to add new classes.
com.acme.model	M	String	A definition of the model. This is usually vendor specific. For example "Mouse".
com.acme.manufacturer	M	String	Manufacturer of this device, for example "ACME Widget Division".
com.acme.revision	O	String	Revision number. For example, "42".
com.acme.serial	O	String	A serial number. For example "SN6751293-12-2112/A".

Table 13 *Example Device Category Properties, M=Mandatory, O=Optional*

11.3.3 Match Example

Driver services and Device services are connected via a matching process that is explained in *The Device Attachment Algorithm* on page 241. The Driver service plays a pivotal role in this matching process. It must inspect the Device service (from its ServiceReference object) that has just been registered and decide if it potentially could cooperate with this Device service.

It must be able to answer a value indicating the quality of the match. The scale of this match value must be defined in the device category so as to allow Driver services to match on a fair basis. The scale must start at least at 1 and go upwards.

Driver services for this sample device category must return one of the match codes defined in the com.acme.widget.WidgetDevice interface or Device.MATCH_NONE if the Device service is not recognized. The device category must define the exact rules for the match codes in the device category specification. In this example, a small range from 2 to 10 (MATCH_NONE is 0) is defined for WidgetDevice devices. They are named in the WidgetDevice interface for convenience and have the following semantics.

Match name	Value	Description
MATCH_SERIAL	10	An exact match, including the serial number.
MATCH_VERSION	8	Matches the right class, make model, and version.
MATCH_MODEL	6	Matches the right class and make model.
MATCH_MAKE	4	Matches the make.
MATCH_CLASS	2	Only matches the class.

Table 14 *Sample Device Category Match Scale*

A Driver service should use the constants to return when it decides how closely the Device service matches its suitability. For example, if it matches the exact serial number, it should return MATCH_SERIAL.

11.4 Driver Services

A Driver service is responsible for attaching to suitable Device services under control of the device manager. Before it can attach a Device service, however, it must compete with other Driver services for control.

If a Driver service wins the competition, it must attach the device in a device category-specific way. After that, it can perform its intended functionality. This functionality is not defined here nor in the device category; this specification only describes the behavior of the Device service, not how the Driver service uses it to implement its intended functionality. A Driver service may register one or more new Device services of another device category or a generic service which models a more refined form of the device.

Both refined Device services as well as generic services should be defined in a Device Category. See *Device Category Specifications* on page 228.

11.4.1 Driver Bundles

A Driver service is, like *all* services, implemented in a bundle, and is recognized by the device manager by registering one or more Driver service objects with the Framework.

Such bundles containing one or more Driver services are called *driver bundles*. The device manager must be aware of the fact that the cardinality of the relationship between bundles and Driver services is 1:1...*.

A driver bundle must register *at least* one Driver service in its BundleActivator.start implementation.

11.4.2 Driver Taxonomy

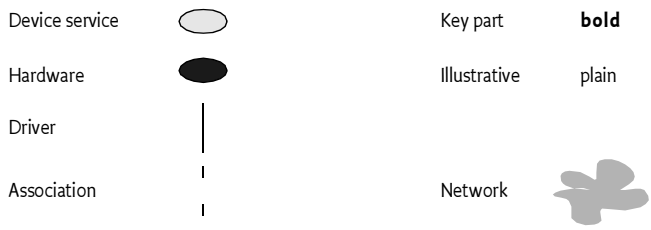
Device Drivers may belong to one of the following categories:

- Base Drivers (Discovery, Pure Discovery and Normal)
- Refining Drivers
- Network Drivers
- Composite Drivers

- Referring Drivers
- Bridging Drivers
- Multiplexing Drivers
- Pure Consuming Drivers

This list is not definitive, and a Driver service is not required to fit into one of these categories. The purpose of this taxonomy is to show the different topologies that have been considered for the Device Access service specification.

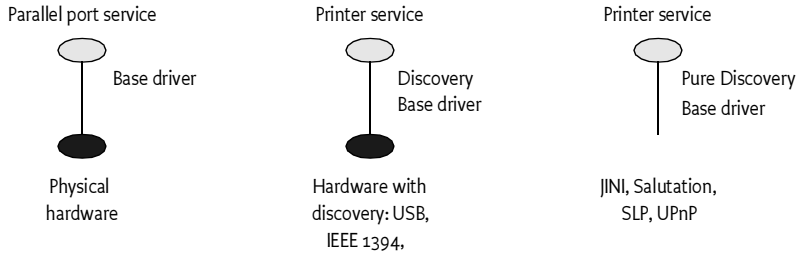
Figure 36 *Legend for Device Driver Services Taxonomy*



11.4.2.1 Base Drivers

The first category of device drivers are called *base drivers* because they provide the lowest-level representation of a physical device. The distinguishing factor is that they are not registered as Driver services because they do not have to compete for access to their underlying technology.

Figure 37 *Base Driver Types*



Base drivers discover physical devices using code not specified here (for example, through notifications from a device driver in native code) and then register corresponding Device services.

When the hardware supports a discovery mechanism and reports a physical device, a Device service is then registered. Drivers supporting a discovery mechanism are called *discovery base drivers*.

An example of a discovery base driver is a USB driver. Discovered USB devices are registered with the Framework as a generic USB Device service. The USB specification (see [29] *USB Specification*) defines a tightly integrated discovery method. Further, devices are individually addressed; no provision exists for broadcasting a message to all devices attached to the USB bus. Therefore, there is no reason to expose the USB network itself; instead, a discovery base driver can register the individual devices as they are discovered.

Not all technologies support a discovery mechanism. For example, most serial ports do not support detection, and it is often not even possible to detect whether a device is attached to a serial port.

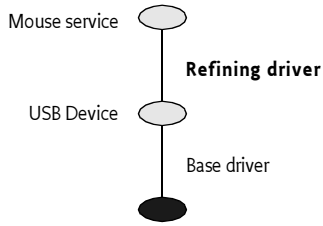
Although each driver bundle should perform discovery on its own, a driver for a non-discoverable serial port requires external help – either through a user interface or by allowing the Configuration Admin service to configure it.

It is possible for the driver bundle to combine automatic discovery of Plug and Play-compliant devices with manual configuration when non-compliant devices are plugged in.

11.4.2.2 Refining Drivers

The second category of device drivers are called *refining drivers*. Refining drivers provide a refined view of a physical device that is already represented by another Device service registered with the Framework. Refining drivers register a Driver service with the Framework. This Driver service is used by the device manager to attach the refining driver to a less refined Device service that is registered as a result of events within the Framework itself.

Figure 38 *Refining Driver Diagram*



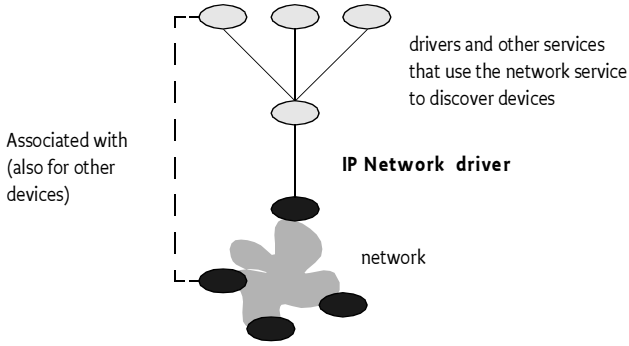
An example of a refining driver is a mouse driver, which is attached to the generic USB Device service representing a physical mouse. It then registers a new Device service which represents it as a Mouse service, defined elsewhere.

The majority of drivers fall into the refining driver type.

11.4.2.3 Network Drivers

An Internet Protocol (IP) capable network such as Ethernet supports individually addressable devices and allows broadcasts, but does not define an intrinsic discovery protocol. In this case, the entire network should be exposed as a single Device service.

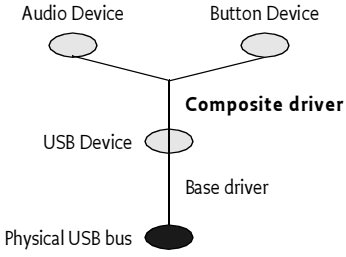
Figure 39 *Network Driver diagram*



11.4.2.4 Composite Drivers

Complex devices can often be broken down into several parts. Drivers that attach to a single service and then register multiple Device services are called *composite drivers*. For example, a USB speaker containing software-accessible buttons can be registered by its driver as two separate Device services: an Audio Device service and a Button Device service.

Figure 40 *Composite Driver structure*



This approach can greatly reduce the number of interfaces needed, as well as enhance reusability.

11.4.2.5 Referring Drivers

A referring driver is actually not a driver in the sense that it controls Device services. Instead, it acts as an intermediary to help locate the correct driver bundle. This process is explained in detail in *The Device Attachment Algorithm* on page 241.

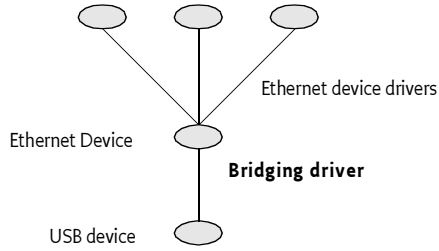
A referring driver implements the call to the attach method to inspect the Device service, and decides which Driver bundle would be able to attach to the device. This process can actually involve connecting to the physical device and communicating with it. The attach method then returns a String object that indicates the DRIVER_ID of another driver bundle. This process is called a referral.

For example, a vendor ACME can implement one driver bundle that specializes in recognizing all of the devices the vendor produces. The referring driver bundle does not contain code to control the device – it contains only sufficient logic to recognize the assortment of devices. This referring driver can be small, yet can still identify a large product line. This approach can drastically reduce the amount of downloading and matching needed to find the correct driver bundle.

11.4.2.6 Bridging Drivers

A bridging driver registers a Device service from one device category but attaches it to a Device service from another device category.

Figure 41 *Bridging Driver Structure*



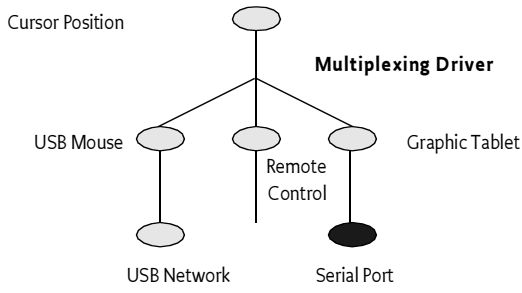
For example, USB to Ethernet bridges exist that allow connection to an Ethernet network through a USB device. In this case, the top level of the USB part of the Device service stack would be an Ethernet Device service. But the same Ethernet Device service can also be the bottom layer of an Ethernet layer of the Device service stack. A few layers up, a bridge could connect into yet another network.

The stacking depth of Device services has no limit, and the same drivers could in fact appear at different levels in the same Device service stack. The graph of drivers-to-Device services roughly mirrors the hardware connections.

11.4.2.7 Multiplexing Drivers

A *multiplexing driver* attaches a number of Device services and aggregates them in a new Device service.

Figure 42 *Multiplexing Driver Structure*

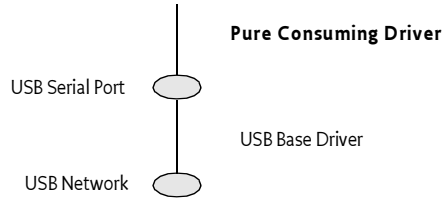


For example, assume that a system has a mouse on USB, a graphic tablet on a serial port, and a remote control facility. Each of these would be registered as a service with the Framework. A multiplexing driver can attach all three, and can merge the different positions in a central Cursor Position service.

11.4.2.8 Pure Consuming Drivers

A *pure consuming driver* bundle will attach to devices without registering a refined version.

Figure 43 *Pure Consuming Driver Structure*



For example, one driver bundle could decide to handle all serial ports through javax.comm instead of registering them as services. When a USB serial port is plugged in, one or more Driver services are attached, resulting in a Device service stack with a Serial Port Device service. A pure consuming driver may then attach to the Serial Port Device service and register a new serial port with the javax.comm.* registry instead of the Framework service registry. This registration effectively transfers the device from the OSGi environment into another environment.

11.4.2.9 Other Driver Types

It should be noted that any bundle installed in the OSGi environment may get and use a Device service without having to register a Driver service.

The following functionality is offered to those bundles that do register a Driver service and conform to the this specification:

- The bundles can be installed and uninstalled on demand.
- Attachment to the Device service is only initiated after the winning the competition with other drivers.

11.4.3 Driver Service Registration

Drivers are recognized by registering a Driver service with the Framework. This event makes the device manager aware of the existence of the Driver service. A Driver service registration must have a DRIVER_ID property whose value is a String object, uniquely identifying the driver to the device manager. The device manager must use the DRIVER_ID to prevent the installation of duplicate copies of the same driver bundle.

Therefore, this DRIVER_ID must:

- Depend only on the specific behavior of the driver, and thus be independent of unrelated aspects like its location or mechanism of downloading.
- Start with the reversed form of the domain name of the company that implements it: for example, com.acme.widget.1.1.

- Differ from the DRIVER_ID of drivers with different behavior. Thus, it must *also* be different for each revision of the same driver bundle so they may be distinguished.

When a new Driver service is registered, the Device Attachment Algorithm must be applied to each idle Device service. This requirement gives the new Driver service a chance to compete with other Driver services for attaching to idle devices. The techniques outlined in *Optimizations* on page 244 can provide significant shortcuts for this situation.

As a result, the Driver service object can receive match and attach requests before the method which registered the service has returned.

This specification does not define any method for new Driver services to *steal* already attached devices. Once a Device service has been attached by a Driver service, it can only be released by the Driver service itself.

11.4.4 Driver Service Unregistration

When a Driver service is unregistered, it must release all Device services to which it is attached. Thus, *all* its attached Device services become idle. The device manager must gather all of these idle Device services and try to re-attach them. This condition gives other Driver services a chance to take over the refinement of devices after the unregistering driver. The techniques outlined in *Optimizations* on page 244 can provide significant shortcuts for this situation.

A Driver service that is installed by the device manager must remain registered as long as the driver bundle is active. Therefore, a Driver service should only be unregistered if the driver bundle is stopping, an occurrence which may precede its being uninstalled or updated. Driver services should thus not unregister in an attempt to minimize resource consumption. Such optimizations can easily introduce race conditions with the device manager.

11.4.5 Driver Service Methods

The Driver interface consists of the following methods:

- match(ServiceReference) – This method is called by the device manager to find out how well this Driver service matches the Device service as indicated by the ServiceReference argument. The value returned here is specific for a device category. If this Device service is of another device category, the value Device.MATCH_NONE must be returned. Higher values indicate a better match. For the exact matching algorithm, see *The Device Attachment Algorithm* on page 241.
  Driver match values and referrals must be deterministic, in that repeated calls for the same Device service must return the same results so that results can be cached by the device manager.
- attach(ServiceReference) – If the device manager decides that a Driver service should be attached to a Device service, it must call this method on the Driver service object. Once this method is called, the Device service is regarded as attached to that Driver service, and no other Driver service must be called to attach to the Device service. The Device service must remain *owned* by the Driver service until the Driver bundle is stopped. No unattach method exists.

The attach method should return null when the Device service is correctly attached. A referring driver (see *Referring Drivers* on page 233) can return a String object that specifies the DRIVER_ID of a driver that can handle this Device service. In this case, the Device service is not attached and the device manager must attempt to install a Driver service with the same DRIVER_ID via a Driver Locator service. The attach method must be deterministic as described in the previous method.

11.4.6 Idle Driver Bundles

An idle Driver bundle is a bundle with a registered Driver service, and is not attached to any Device service. Idle Driver bundles are consuming resources in the OSGi Service Platform. The device manager should uninstall bundles that it has installed and which are idle.

11.5 Driver Locator Service

The device manager must automatically install Driver bundles, which are obtained from Driver Locator services, when new Device services are registered.

A Driver Locator service encapsulates the knowledge of how to fetch the Driver bundles needed for a specific Device service. This selection is made on the properties that are registered with a device: for example, DEVICE_CATEGORY and any other properties registered with the Device service registration.

The purpose of the Driver Locator service is to separate the mechanism from the policy. The decision to install a new bundle is made by the device manager (the mechanism), but a Driver Locator service decides which bundle to install and from where the bundle is downloaded (the policy).

Installing bundles has many consequences for the security of the system, and this process is also sensitive to network setup and other configuration details. Using Driver Locator services allows the Operator to choose a strategy that best fits its needs.

Driver services are identified by the DRIVER_ID property. Driver Locator services use this particular ID to identify the bundles that can be installed. Driver ID properties have uniqueness requirements as specified in *Device Service Registration* on page 226. This uniqueness allows the device manager to maintain a list of Driver services and prevent unnecessary installs.

An OSGi Service Platform can have several different Driver Locator services installed. The device manager must consult all of them and use the combined result set, after pruning duplicates based on the DRIVER_ID values.

11.5.1 The DriverLocator Interface

The DriverLocator interface allows suitable driver bundles to be located, downloaded, and installed on demand, even when completely unknown devices are detected.

It has the following methods:

- findDrivers(Dictionary) – This method returns an array of driver IDs that potentially match a service described by the properties in the Dictionary object. A driver ID is the String object that is registered by a Driver service under the DRIVER_ID property.
- loadDriver(String) – This method returns an InputStream object that can be used to download the bundle containing the Driver service as specified by the driver ID argument. If the Driver Locator service cannot download such a bundle, it should return null. Once this bundle is downloaded and installed in the Framework, it must register a Driver service with the DRIVER_ID property set to the value of the String argument.

11.5.2 A Driver Example

The following example shows a very minimal Driver service implementation. It consists of two classes. The first class is SerialWidget. This class tracks a single WidgetDevice from *Sample Device Category Specification* on page 228. It registers a javax.comm.SerialPort service, which is a general serial port specification that could also be implemented from other device categories like USB, a COM port, etc. It is created when the SerialWidgetDriver object is requested to attach a WidgetDevice by the device manager. It registers a new javax.comm.SerialPort service in its constructor.

The org.osgi.util.tracker.ServiceTracker is extended to handle the Framework events that are needed to simplify tracking this service. The removedService method of this class is overridden to unregister the SerialPort when the underlying WidgetDevice is unregistered.

```
package com.acme.widget;
import org.osgi.service.device.*;
import org.osgi.framework.*;
import org.osgi.util.tracker.*;

class SerialWidget extends ServiceTracker
   implements javax.comm.SerialPort,
      org.osgi.service.device.Constants {
   ServiceRegistration        registration;

   SerialWidget( BundleContext c, ServiceReference r ) {
      super( c, r, null );
      open();
   }

   public Object addingService( ServiceReference ref ) {
      WidgetDevice dev = (WidgetDevice)
         context.getService( ref );
      registration = context.registerService(
           javax.comm.SerialPort.class.getName(),
           this,
           null );
         return dev;
   }

   public void removedService( ServiceReference ref,
```

```
        Object service ) {
        registration.unregister();
        context.ungetService(ref);
    }
    ... methods for javax.comm.SerialPort that are
    ... converted to underlying WidgetDevice
}
```

A SerialWidgetDriver object is registered with the Framework in the Bundle Activator start method under the Driver interface. The device manager must call the match method for each idle Device service that is registered. If it is chosen by the device manager to control this Device service, a new SerialWidget is created that offers serial port functionality to other bundles.

```
public class SerialWidgetDriver implements Driver {
    BundleContext           context;

    String     spec =
          "(&"
        +" (objectclass=com.acme.widget.WidgetDevice)"
        +"  (DEVICE_CATEGORY=WidgetDevice)"
        +"  (com.acme.class=Serial)"
        + ")";

    Filter     filter;

    SerialWidgetDriver( BundleContext context )
        throws Exception {
        this.context = context;
        filter = context.createFilter(spec);
    }
    public int match( ServiceReference d ) {
        if ( filter.match( d ) )
            return WidgetDevice.MATCH_CLASS;
        else
            return Device.MATCH_NONE;
    }
    public synchronized String attach(ServiceReference r){
        new SerialWidget( context, r );
    }
}
```

11.6 The Driver Selector Service

The purpose of the Driver Selector service is to customize the selection of the best Driver service from a set of suitable Driver bundles. The device manager has a default algorithm as described in *The Device Attachment Algorithm* on page 241. When this algorithm is not sufficient and requires customizing by the operator, a bundle providing a Driver Selector service can be installed in the Framework. This service must be used by the device manager as the final arbiter when selecting the best match for a Device service.

The Driver Selector service is a singleton; only one such service is recognized by the device manager. The Framework method BundleContext.getServiceReference must be used to obtain a Driver Selector service. In the erroneous case that multiple Driver Selector services are registered, the service.ranking property will thus define which service is actually used.

A device manager implementation must invoke the method select(ServiceReference,Match[]). This method receives a Service Reference to the Device service and an array of Match objects. Each Match object contains a link to the ServiceReference object of a Driver service and the result of the match value returned from a previous call to Driver.match. The Driver Selector service should inspect the array of Match objects and use some means to decide which Driver service is best suited. The index of the best match should be returned. If none of the Match objects describe a possible Driver service, the implementation must return DriverSelector.SELECT_NONE (-1).

11.7 Device Manager

Device Access is controlled by the device manager in the background. The device manager is responsible for initiating all actions in response to the registration, modification, and unregistration of Device services and Driver services, using Driver Locator services and a Driver Selector service as helpers.

The device manager detects the registration of Device services and coordinates their attachment with a suitable Driver service. Potential Driver services do not have to be active in the Framework to be eligible. The device manager must use Driver Locator services to find bundles that might be suitable for the detected Device service and that are not currently installed. This selection is done via a DRIVER_ID property that is unique for each Driver service.

The device manager must install and start these bundles with the help of a Driver Locator service. This activity must result in the registration of one or more Driver services. All available Driver services, installed by the device manager and also others, then participate in a bidding process. The Driver service can inspect the Device service through its ServiceReference object to find out how well this Driver service matches the Device service.

If a Driver Selector service is available in the Framework service registry, it is used to decide which of the eligible Driver services is the best match.

If no Driver Selector service is available, the highest bidder must win, with tie breaks defined on the service.ranking and service.id properties. The selected Driver service is then asked to attach the Device service.

If no Driver service is suitable, the Device service remains idle. When new Driver bundles are installed, these idle Device services must be reattached.

The device manager must reattach a Device service if, at a later time, a Driver service is unregistered due to an uninstallation or update. At the same time, however, it should prevent superfluous and non-optimal reattachments. The device manager should also garbage-collect driver bundles it installed which are no longer used.

The device manager is a singleton. Only one device manager may exist, and it must have no public interface.

11.7.1 Device Manager Startup

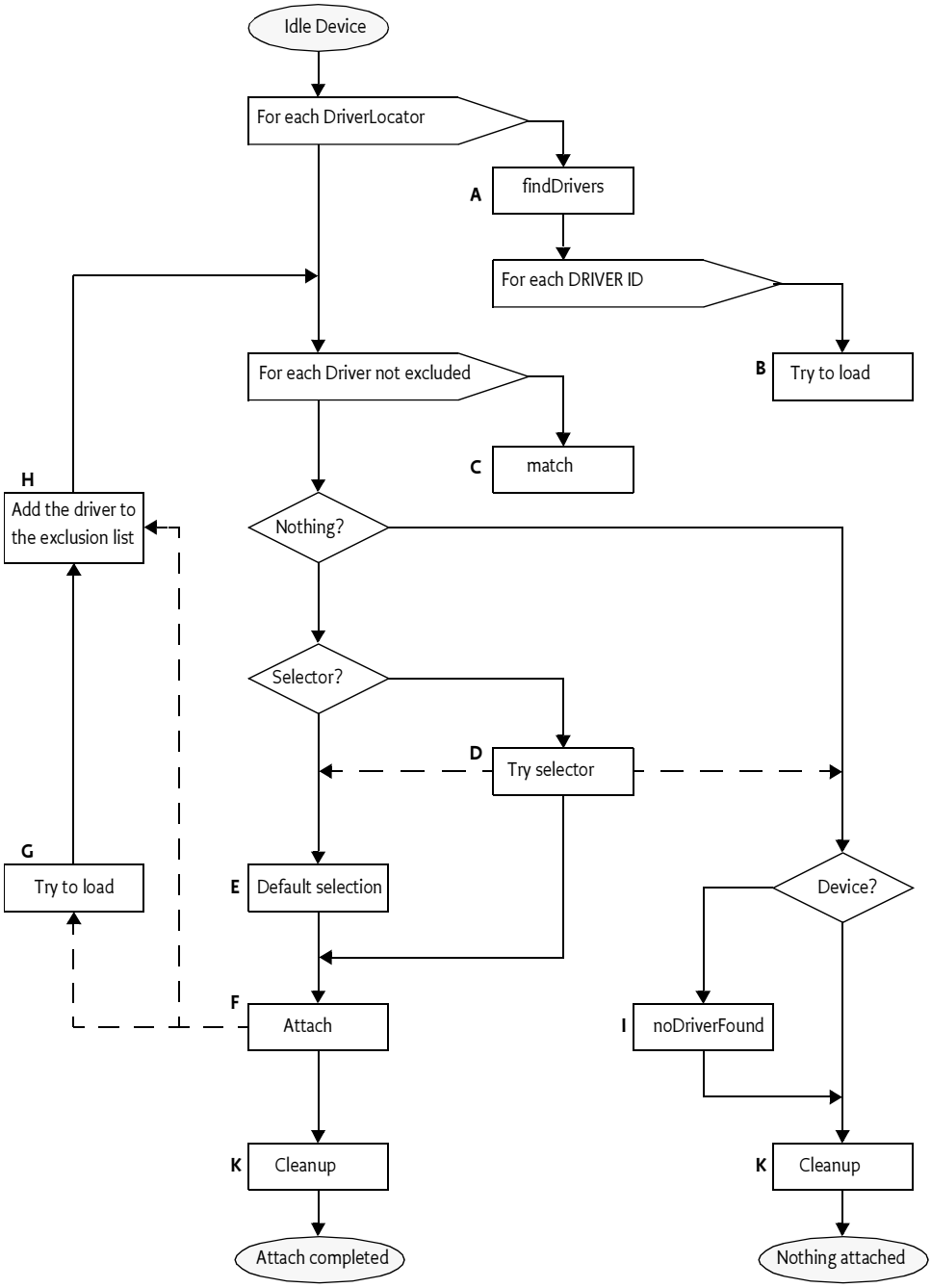
To prevent race conditions during Framework startup, the device manager must monitor the state of Device services and Driver services immediately when it is started. The device manager must not, however, begin attaching Device services until the Framework has been fully started, to prevent superfluous or non-optimal attachments.

The Framework has completed starting when the FrameworkEvent.STARTED event has been published. Publication of that event indicates that Framework has finished all its initialization and all bundles are started. If the device manager is started after the Framework has been initialized, it should detect the state of the Framework by examining the state of the system bundle.

11.7.2 The Device Attachment Algorithm

A key responsibility of the device manager is to attach refining drivers to idle devices. The following diagram illustrates the device attachment algorithm.

Figure 44 Device Attachment Algorithm



11.7.3 Legend

Step	Description
A	DriverLocator.findDrivers is called for each registered Driver Locator service, passing the properties of the newly detected Device service. Each method call returns zero or more DRIVER_ID values (identifiers of particular driver bundles). If the findDrivers method throws an exception, it is ignored, and processing continues with the next Driver Locator service. See *Optimizations* on page 244 for further guidance on handling exceptions.
B	For each found DRIVER_ID that does not correspond to an already registered Driver service, the device manager calls DriverLocator.loadDriver to return an InputStream containing the driver bundle. Each call to loadDriver is directed to one of the Driver Locator services that mentioned the DRIVER_ID in step A. If the loadDriver method fails, the other Driver Locator objects are tried. If they all fail, the driver bundle is ignored. If this method succeeds, the device manager installs and starts the driver bundle. Driver bundles must register their Driver services synchronously during bundle activation.
C	For each Driver service, except those on the exclusion list, call its Driver.match method, passing the ServiceReference object to the Device service. Collect all successful matches – that is, those whose return values are greater than Device.MATCH_NONE – in a list of active matches. A match call that throws an exception is considered unsuccessful and is not added to the list.
D	If there is a Driver Selector service, the device manager calls the DriverSelector.select method, passing the array of active Match objects. If the Driver Selector service returns the index of one of the Match objects from the array, its associated Driver service is selected for attaching the Device service. If the Driver Selector service returns DriverSelector.SELECT_NONE, no Driver service must be considered for attaching the Device service. If the Driver Selector service throws an exception or returns an invalid result, the default selection algorithm is used. Only one Driver Selector service is used, even if there is more than one registered in the Framework. See *The Driver Selector Service* on page 239.
E	The winner is the one with the highest match value. Tie breakers are respectively: • Highest service.ranking property. • Lowest service.id property.

Table 15 *Driver attachment algorithm*

Step	Description
F	The selected Driver service's attach method is called. If the attach method returns null, the Device service has been successfully attached. If the attach method returns a String object, it is interpreted as a referral to another Driver service and processing continues at G. See *Referring Drivers* on page 233. If an exception is thrown, the Driver service has failed, and the algorithm proceeds to try another Driver service after excluding this one from further consideration at Step H.
G	The device manager attempts to load the referred driver bundle in a manner similar to Step B, except that it is unknown which Driver Locator service to use. Therefore, the loadDriver method must be called on each Driver Locator service until one succeeds (or they all fail). If one succeeds, the device manager installs and starts the driver bundle. The driver bundle must register a Driver service during its activation which must be added to the list of Driver services in this algorithm.
H	The referring driver bundle is added to the exclusion list. Because each new referral adds an entry to the exclusion list, which in turn disqualifies another driver from further matching, the algorithm cannot loop indefinitely. This list is maintained for the duration of this algorithm. The next time a new Device service is processed, the exclusion list starts out empty.
I	If no Driver service attached the Device service, the Device service is checked to see whether it implements the Device interface. If so, the noDriverFound method is called. Note that this action may cause the Device service to unregister and possibly a new Device service (or services) to be registered in its place. Each new Device service registration must restart the algorithm from the beginning.
K	Whether an attachment was successful or not, the algorithm may have installed a number of driver bundles. The device manager should remove any idle driver bundles that it installed.

Table 15 *Driver attachment algorithm*

11.7.4 Optimizations

Optimizations are explicitly allowed and even recommended for an implementation of a device manager. Implementations may use the following assumptions:

- Driver match values and referrals must be deterministic, in that repeated calls for the same Device service must return the same results.
- The device manager may cache match values and referrals. Therefore, optimizations in the device attachment algorithm based on this assumption are allowed.
- The device manager may delay loading a driver bundle until it is needed. For example, a delay could occur when that DRIVER_ID's match values are cached.

- The results of calls to DriverLocator and DriverSelector methods are not required to be deterministic, and must not be cached by the device manager.
- Thrown exceptions must not be cached. Exceptions are considered transient failures, and the device manager must always retry a method call even if it has thrown an exception on a previous invocation with the same arguments.

11.7.5 Driver Bundle Reclamation

The device manager may remove driver bundles it has installed at any time, provided that all the Driver services in that bundle are idle. This recommended practice prevents unused driver bundles from accumulating over time. Removing driver bundles too soon, however, may cause unnecessary installs and associated delays when driver bundles are needed again.

If a device manager implements driver bundle reclamation, the specified matching algorithm is not guaranteed to terminate unless the device manager takes reclamation into account.

For example, assume that a new Device service triggers the attachment algorithm. A driver bundle recommended by a Driver Locator service is loaded. It does not match, so the Device service remains idle. The device manager is eager to reclaim space, and unloads the driver bundle. The disappearance of the Driver service causes the device manager to reattach idle devices. Because it has not kept a record of its previous activities, it tries to reattach the same device, which closes the loop.

On systems where the device manager implements driver bundle reclamation, all refining drivers should be loaded through Driver Locator services. This recommendation is intended to prevent the device manager from erroneously uninstalling pre-installed driver bundles that cannot later be reinstalled when needed.

The device manager can be updated or restarted. It cannot, however, rely on previously stored information to determine which driver bundles were preinstalled and which were dynamically installed and thus are eligible for removal. The device manager may persistently store cachable information for optimization, but must be able to cold start without any persistent information and still be able to manage an existing connection state, satisfying all of the requirements in this specification.

11.7.6 Handling Driver Bundle Updates

It is not straightforward to determine whether a driver bundle is being updated when the UNREGISTER event for a Driver service is received. In order to facilitate this distinction, the device manager should wait for a period of time after the unregistration for one of the following events to occur:

- A BundleEvent.UNINSTALLED event for the driver bundle.
- A ServiceEvent.REGISTERED event for another Driver service registered by the driver bundle.

If the driver bundle is uninstalled, or if neither of the above events are received within the allotted time period, the driver is assumed to be inactive. The appropriate waiting period is implementation-dependent and will vary for different installations. As a general rule, this period should be long enough to allow a driver to be stopped, updated, and restarted under normal conditions, and short enough not to cause unnecessary delays in reattaching devices. The actual time should be configurable.

11.7.7 Simultaneous Device Service and Driver Service Registration

The device attachment algorithm may cause driver bundles to be installed, which requires executing the device attachment algorithm recursively. In this case, the appearance of the new driver bundles should be queued until completion of the device attachment algorithm.

Only one device attachment algorithm may be in progress at any moment in time.

The following example sequence illustrates this process when a Driver service is registered:

- Collect the set of all idle devices.
- Apply the device attachment algorithm to each device in the set.
- If no Driver services were registered during the execution of the device attachment algorithm, processing terminates.
- Otherwise, restart this process.

11.8 Security

The device manager is the only privileged bundle in the Device Access specification and requires the org.osgi.AdminPermission to install and uninstall driver bundles.

The device manager itself should be free from any knowledge of policies and should not actively set bundle permissions. Rather, if permissions must be set, it is up to the Management Agent to listen to synchronous bundle events and set the appropriate permissions.

Driver Locator services can trigger the download of any bundle, because they deliver the content of a bundle to the privileged device manager and could potentially insert a Trojan horse into the environment. Therefore, Driver Locator bundles need the ServicePermission[REGISTER, DriverLocator] to register Driver Locator services, and the operator should exercise prudence in assigning this ServicePermission.

Bundles with Driver Selector services only require ServicePermission[REGISTER,DriverSelector] to register the DriverSelector service. The Driver Selector service can play a crucial role in the selection of a suitable Driver service, but it has no means to define a specific bundle itself.

11.9 Changes

The Device Access specification has not increased its version number because no API change has been necessary. The only change to this specification has been a clarification of the concept of an idle device.

11.10 org.osgi.service.device

The OSGi Device Access 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.device; specification-version=1.1
```

11.10.1 Summary

- Constants – This interface defines standard names for property keys associated with Device[p.248] and Driver[p.248] services. [p.247]
- Device – Interface for identifying device services.[p.248]
- Driver – A `Driver` service object must be registered by each Driver bundle wishing to attach to Device services provided by other drivers. [p.248]
- DriverLocator – A Driver Locator service can find and load device driver bundles given a property set. [p.250]
- DriverSelector – When the device manager detects a new Device service, it calls all registered Driver services to determine if anyone matches the Device service. [p.250]
- Match – Instances of `Match` are used in the DriverSelector.select[p.250] method to identify Driver services matching a Device service. [p.251]

11.10.2 public interface Constants

This interface defines standard names for property keys associated with Device[p.248] and Driver[p.248] services.

The values associated with these keys are of type `java.lang.String`, unless otherwise stated.

See Also `Device`[p.248], `Driver`[p.248]

Since 1.1

11.10.2.1 public static final String DEVICE_CATEGORY = "DEVICE_CATEGORY"

Property (named "DEVICE_CATEGORY") containing a human readable description of the device categories implemented by a device. This property is of type `String[]`

Services registered with this property will be treated as devices and discovered by the device manager

11.10.2.2 public static final String DEVICE_DESCRIPTION =

"DEVICE_DESCRIPTION"

Property (named "DEVICE_DESCRIPTION") containing a human readable string describing the actual hardware device.

11.10.2.3 public static final String DEVICE_SERIAL = "DEVICE_SERIAL"

Property (named "DEVICE_SERIAL") specifying a device's serial number.

11.10.2.4 public static final String DRIVER_ID = "DRIVER_ID"

Property (named "DRIVER_ID") identifying a driver.

A DRIVER_ID should start with the reversed domain name of the company that implemented the driver (e.g., com.acme), and must meet the following requirements:

- It must be independent of the location from where it is obtained.
- It must be independent of the DriverLocator[p.250] service that downloaded it.
- It must be unique.
- It must be different for different revisions of the same driver.

This property is mandatory, i.e., every Driver service must be registered with it.

11.10.3 public interface Device

Interface for identifying device services.

A service must implement this interface or use the Constants.DEVICE_CATEGORY[p.247] registration property to indicate that it is a device. Any services implementing this interface or registered with the DEVICE_CATEGORY property will be discovered by the device manager.

Device services implementing this interface give the device manager the opportunity to indicate to the device that no drivers were found that could (further) refine it. In this case, the device manager calls the noDriverFound[p.248] method on the Device object.

Specialized device implementations will extend this interface by adding methods appropriate to their device category to it.

See Also Driver[p.248]

11.10.3.1 public static final int MATCH_NONE = 0

Return value from Driver.match[p.249] indicating that the driver cannot refine the device presented to it by the device manager. The value is zero.

11.10.3.2 public void noDriverFound( )

□ Indicates to this Device object that the device manager has failed to attach any drivers to it.

If this Device object can be configured differently, the driver that registered this Device object may unregister it and register a different Device service instead.

11.10.4 public interface Driver

A Driver service object must be registered by each Driver bundle wishing to attach to Device services provided by other drivers. For each newly discovered Device[p.248] object, the device manager enters a bidding phase. The Driver object whose match[p.249] method bids the highest for a particular Device object will be instructed by the device manager to attach to the Device object.

See Also Device[p.248], DriverLocator[p.250]

11.10.4.1 public String attach(ServiceReference reference) throws Exception

reference the ServiceReference object of the device to attach to

- Attaches this Driver service to the Device service represented by the given ServiceReference object.

  A return value of null indicates that this Driver service has successfully attached to the given Device service. If this Driver service is unable to attach to the given Device service, but knows of a more suitable Driver service, it must return the DRIVER_ID of that Driver service. This allows for the implementation of referring drivers whose only purpose is to refer to other drivers capable of handling a given Device service.

  After having attached to the Device service, this driver may register the underlying device as a new service exposing driver-specific functionality.

  This method is called by the device manager.

Returns null if this Driver service has successfully attached to the given Device service, or the DRIVER_ID of a more suitable driver

Throws Exception – if the driver cannot attach to the given device and does not know of a more suitable driver

11.10.4.2 public int match(ServiceReference reference) throws Exception

reference the ServiceReference object of the device to match

- Checks whether this Driver service can be attached to the Device service. The Device service is represented by the given ServiceReference and returns a value indicating how well this driver can support the given Device service, or Device.MATCH_NONE[p.248] if it cannot support the given Device service at all.

  The return value must be one of the possible match values defined in the device category definition for the given Device service, or Device.MATCH_NONE if the category of the Device service is not recognized.

  In order to make its decision, this Driver service may examine the properties associated with the given Device service, or may get the referenced service object (representing the actual physical device) to talk to it, as long as it ungets the service and returns the physical device to a normal state before this method returns.

  A Driver service must always return the same match code whenever it is presented with the same Device service.

  The match function is called by the device manager during the matching process.

Returns value indicating how well this driver can support the given Device service, or Device.MATCH_NONE if it cannot support the Device service at all

Throws Exception – if this Driver service cannot examine the Device service

11.10.5 public interface DriverLocator

A Driver Locator service can find and load device driver bundles given a property set. Each driver is represented by a unique DRIVER_ID.

Driver Locator services provide the mechanism for dynamically downloading new device driver bundles into an OSGi environment. They are supplied by providers and encapsulate all provider-specific details related to the location and acquisition of driver bundles.

See Also Driver[p.248]

11.10.5.1 public String[] findDrivers(Dictionary props)

props the properties of the device for which a driver is sought

- Returns an array of DRIVER_ID strings of drivers capable of attaching to a device with the given properties.

  The property keys in the specified Dictionary objects are case-insensitive.

Returns array of driver DRIVER_ID strings of drivers capable of attaching to a Device service with the given properties, or null if this Driver Locator service does not know of any such drivers

11.10.5.2 public InputStream loadDriver(String id) throws IOException

id the DRIVER_ID of the driver that needs to be installed.

- Get an InputStream from which the driver bundle providing a driver with the giving DRIVER_ID can be installed.

Returns a InputStream object from which the driver bundle can be installed

Throws IOException – the input stream for the bundle cannot be created

11.10.6 public interface DriverSelector

When the device manager detects a new Device service, it calls all registered Driver services to determine if anyone matches the Device service. If at least one Driver service matches, the device manager must choose one. If there is a Driver Selector service registered with the Framework, the device manager will ask it to make the selection. If there is no Driver Selector service, or if it returns an invalid result, or throws an Exception, the device manager uses the default selection strategy.

Since 1.1

11.10.6.1 public static final int SELECT_NONE = -1

Return value from DriverSelector.select, if no Driver service should be attached to the Device service. The value is -1.

11.10.6.2 public int select(ServiceReference reference, Match[] matches)

reference the ServiceReference object of the Device service.

matches the array of all non-zero matches.

- Select one of the matching Driver services. The device manager calls this method if there is at least one driver bidding for a device. Only Driver services that have responded with nonzero (not Device.MATCH_NONE[p.248]) match values will be included in the list.

Returns index into the array of Match objects, or SELECT_NONE if no Driver service should be attached

11.10.7 public interface Match

Instances of Match are used in the DriverSelector.select[p.250] method to identify Driver services matching a Device service.

See Also DriverSelector[p.250]

Since 1.1

11.10.7.1 public ServiceReference getDriver( )

- Return the reference to a Driver service.

Returns ServiceReference object to a Driver service.

11.10.7.2 public int getMatchValue( )

- Return the match value of this object.

Returns the match value returned by this Driver service.

11.11 References

[28] *Java Communications API*
http://java.sun.com/products/javacomm

[29] *USB Specification*
http://www.usb.org/developers/data/usbspec.zip

[30] *Universal Plug and Play*
http://www.upnp.org

[31] *Jini, Service Discovery and Usage*
http://www.jini.org/resources/

[32] *Salutation, Service Discovery Protocol*
http://www.salutation.org

12 User Admin Service Specification

Version 1.0

12.1 Introduction

OSGi Service Platforms are often used in places where end users or devices initiate actions. These kinds of actions inevitably create a need for authenticating the initiator. Authenticating can be done in many different ways, including with passwords, one-time token cards, bio-metrics, and certificates.

Once the initiator is authenticated, it is necessary to verify that this principal is authorized to perform the requested action. This authorization can only be decided by the operator of the OSGi environment, and thus requires administration.

The User Admin service provides this type of functionality. Bundles can use the User Admin service to authenticate an initiator and represent this authentication as an Authorization object. Bundles that execute actions on behalf of this user can use the Authorization object to verify if that user is authorized.

The User Admin service provides authorization based on who runs the code, instead of using the Java code-based permission model. See [33] *The Java Security Architecture for JDK 1.2*. It performs a role similar to [34] *Java Authentication and Authorization Service*.

12.1.1 Essentials

- *Authentication* – A large number of authentication schemes already exist, and more will e developed. The User Admin service must be flexible enough to adapt to the many different authentication schemes that can be run on a computer system.
- *Authorization* – All bundles should use the User Admin service to authenticate users and to find out if those users are authorized. It is therefore paramount that a bundle can find out authorization information with little effort.
- *Security* – Detailed security, based on the Framework security model, is needed to provide safe access to the User Admin service. It should allow limited access to the credentials and other properties.
- *Extensibility* – Other bundles should be able to build on the User Admin service. It should be possible to examine the information from this service and get real-time notifications of changes.
- *Properties* – The User Admin service must maintain a persistent database of users. It must be possible to use this database to hold more information about this user.

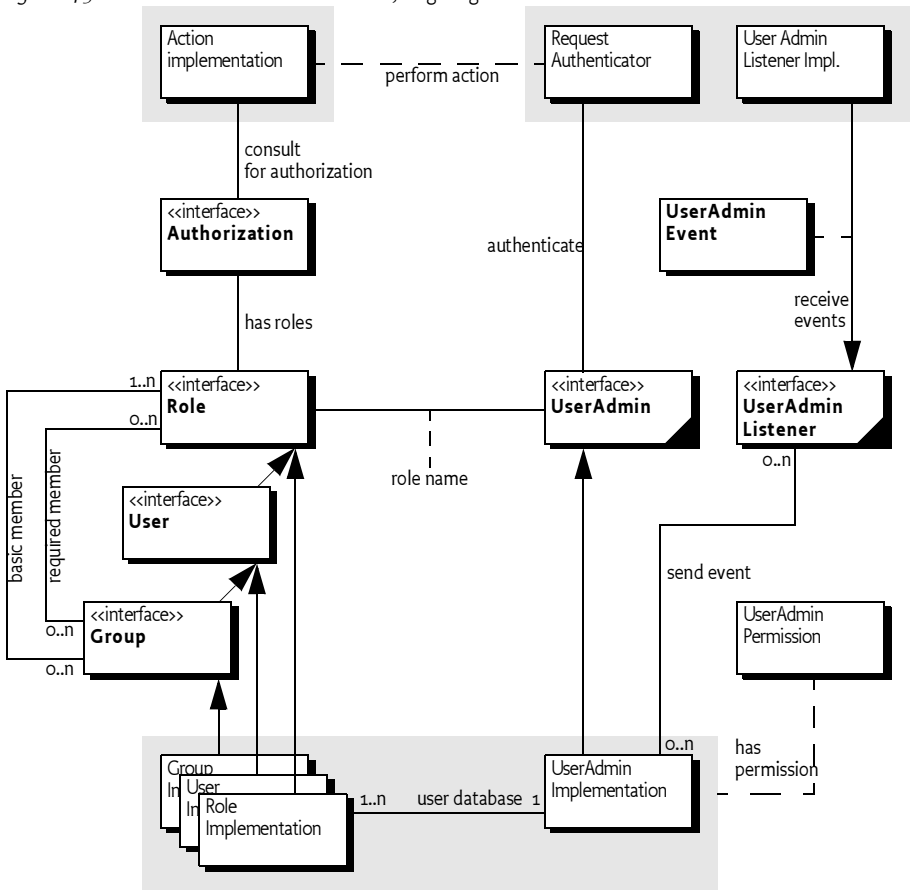
- *Administration* – Administering authorizations for each possible action and initiator is time-consuming and error-prone. It is therefore necessary to have mechanisms to group end users and make it simple to assign authorizations to all members of a group at one time.

12.1.2 Entities

This Specification defines the following User Admin service entities:

- *UserAdmin* – This interface manages a database of named roles which can be used for authorization and authentication purposes.
- *Role* – This interface exposes the characteristics shared by all roles: a name, a type, and a set of properties.
- *User* – This interface (which extends Role) is used to represent any entity which may have credentials associated with it. These credentials can be used to authenticate an initiator.
- *Group* – This interface (which extends User) is used to contain an aggregation of named Role objects (Group or User objects).
- *Authorization* – This interface encapsulates an authorization context on which bundles can base authorization decisions.
- *UserAdminEvent* – This class is used to represent a role change event.
- *UserAdminListener* – This interface provides a listener for events of type UserAdminEvent that can be registered as a service.
- *UserAdminPermission* – This permission is needed to configure and access the roles managed by a User Admin service.

Figure 45 *User Admin Service*, org.osgi.service.useradmin



12.1.3 Operation

An Operator uses the User Admin service to define OSGi Service Platform users and configure them with properties, credentials, and *roles*.

A Role object represents the initiator of a request (human or otherwise). This specification defines two types of roles:

- *User* – A User object can be configured with credentials, such as a password, and properties, such as address, telephone number, and so on.
- *Group* – A Group object is an aggregation of *basic* and *required* roles. Basic and required roles are used in the authorization phase.

An OSGi Service Platform can have several entry points, each of which will be responsible for authenticating incoming requests. An example of an entry point is the Http Service, which delegates authentication of incoming requests to the handleSecurity method of the HttpContext object that was specified when the target servlet or resource of the request was registered.

The OSGi Service Platform entry points should use the information in the User Admin service to authenticate incoming requests, such as a password stored in the private credentials or the use of a certificate.

A bundle can determine if a request for an action is authorized by looking for a Role object that has the name of the requested action.

The bundle may execute the action if the Role object representing the initiator *implies* the Role object representing the requested action.

For example, an initiator Role object *X* implies an action Group object *A* if:

- *X* implies at least one of *A*'s basic members, and
- *X* implies all of *A*'s required members.

An initiator Role object *X* implies an action User object *A* if:

- *A* and *X* are equal.

The Authorization class handles this non-trivial logic. The User Admin service can capture the privileges of an authenticated User object into an Authorization object. The Authorization.hasRole method checks if the authenticate User object has (or implies) a specified action Role object.

For example, in the case of the Http Service, the HttpContext object can authenticate the initiator and place an Authorization object in the request header. The servlet calls the hasRole method on this Authorization object to verify that the initiator has the authority to perform a certain action. See *Authentication* on page 295.

12.2 Authentication

The authentication phase determines if the initiator is actually the one it says it is. Mechanisms to authenticate always need some information related to the user or the OSGi Service Platform to authenticate an external user. This information can consist of the following:

- A secret known only to the initiator.
- Knowledge about cards that can generate a unique token.
- Public information like certificates of trusted signers.
- Information about the user that can be measured in a trusted way.
- Other specific information.

12.2.1 Repository

The User Admin service offers a repository of Role objects. Each Role object has a unique name and a set of properties that are readable by anyone, and are changeable when the changer has the UserAdminPermission. Additionally, User objects, a sub-interface of Role, also have a set of private protected properties called credentials. Credentials are an extra set of properties that are used to authenticate users and that are protected by UserAdminPermission.

Properties are accessed with the Role.getProperties() method and credentials with the User.getCredentials() method. Both methods return a Dictionary object containing key/value pairs. The keys are String objects and the values of the Dictionary object are limited to String or byte[ ] objects.

This specification does not define any standard keys for the properties or credentials. The keys depend on the implementation of the authentication mechanism and are not formally defined by OSGi specifications.

The repository can be searched for objects that have a unique property (key/value pair) with the method UserAdmin.getUser(String,String). This makes it easy to find a specific user related to a specific authentication mechanism. For example, a secure card mechanism that generates unique tokens could have a serial number identifying the user. The owner of the card could be found with the method

```
User owner = useradmin.getUser(
   "secure-card-serial", "132456712-1212" );
```

If multiple User objects have the same property (key *and* value), a null is returned.

There is a convenience method to verify that a user has a credential without actually getting the credential. This is the User.hasCredential(String, Object) method.

Access to credentials is protected on a name basis by UserAdminPermission. Because properties can be read by anyone with access to a User object, UserAdminPermission only protects change access to properties.

12.2.2 Basic Authentication

The following example shows a very simple authentication algorithm based on passwords.

The vendor of the authentication bundle uses the property "com.acme.basic-id" to contain the name of a user as it logs in. This property is used to locate the User object in the repository. Next, the credential "com.acme.password" contains the password and is compared to the entered password. If the password is correct, the User object is returned. In all other cases a SecurityException is thrown.

```
public User authenticate(
     UserAdmin ua, String name, String pwd )
   throws SecurityException {
   User user = ua.getUser("com.acme.basicid",
      username);
   if (user == null)
      throw new SecurityException( "No such user" );

   if (!user.hasCredential("com.acme.password", pwd))
      throw new SecurityException(
         "Invalid password" );
   return user;
}
```

12.2.3 Certificates

Authentication based on certificates does not require a shared secret. Instead, a certificate contains a name, a public key, and the signature of one or more signers.

The name in the certificate can be used to locate a User object in the repository. Locating a User object, however, only identifies the initiator and does not authenticate it.

1. The first step to authenticate the initiator is to verify that it has the private key of the certificate.
2. Next, the User Admin service must verify that it has a User object with the right property, for example "com.acme.certificate"="Fudd".
3. The next step is to see if the certificate is signed by a trusted source. The bundle could use a central list of trusted signers and only accept certificates signed by those sources. Alternatively, it could require that the certificate itself is already stored in the repository under a unique key as a byte[] in the credentials.
4. In any case, once the certificate is verified, the associated User object is authenticated.

12.3 Authorization

The User Admin service authorization architecture is a *role-based model*. In this model, every action that can be performed by a bundle is associated with a *role*. Such a role is a Group object (called group from now on) from the User Admin service repository. For example, if a servlet could be used to activate the alarm system, there should be a group named AlarmSystemActivation.

The operator can administrate authorizations by populating the group with User objects (users) and other groups. Groups are used to minimize the amount of administration required. For example, it is easier to create one Administrators group and add administrative roles to it rather than individually administer all users for each role. Such a group requires only one action to remove or add a user as an administrator.

The authorization decision can now be made in two fundamentally different ways:

An initiator could be allowed to carry out an action (represented by a Group object) if it implied any of the Group object's members. For example, the AlarmSystemActivation Group object contains an Administrators and a Family Group object:

```
Administrators            = { Elmer, Pepe, Bugs }
Family                    = { Elmer, Pepe, Daffy }

AlarmSystemActivation     = { Administrators, Family }
```

Any of the four members Elmer, Pepe, Daffy, or Bugs can activate the alarm system.

Alternatively, an initiator could be allowed to perform an action (represented by a Group object) if it implied *all* the Group object's members. In this case, using the same AlarmSystemActivation group, only Elmer and Pepe would be authorized to activate the alarm system, since Daffy and Bugs are *not* members of *both* the Administrators and Family Group objects.

The User Admin service supports a combination of both strategies by defining both a set of *basic members* (any) and a set of *required members* (all).

```
Administrators  = { Elmer, Pepe, Bugs }
Family          = { Elmer, Pepe, Daffy }

AlarmSystemActivation
   required     = { Administrators }
   basic        = { Family }
```

The difference is made when Role objects are added to the Group object. To add a basic member, use the Group.addMember(Role) method. To add a required member, use the Group.addRequiredMember(Role) method.

Basic members define the set of members that can get access and required members reduce this set by requiring the initiator to *imply* each required member.

A User object implies a Group object if it implies the following:

- *All* of the Group's required members, and
- At *least* one of the Group's basic members

A User object always implies itself.

If only required members are used to qualify the implication, then the standard user user.anyone can be obtained from the User Admin service and added to the Group object. This Role object is implied by anybody and therefore does not affect the required members.

12.3.1 The Authorization Object

The complexity of authorization is hidden in an Authorization class. Normally, the authenticator should retrieve an Authorization object from the User Admin service by passing the authenticated User object as an argument. This Authorization object is then passed to the bundle that performs the action. This bundle checks the authorization with the Authorization.hasRole(String) method. The performing bundle must pass the name of the action as an argument. The Authorization object checks whether the authenticated user implies the Role object, specifically a Group object, with the given name. This is shown in the following example.

```
public void activateAlarm(Authorization auth) {
   if ( auth.hasRole( "AlarmSystemActivation" ) ) {
      // activate the alarm
      ...
   }
   else throw new SecurityException(
      "Not authorized to activate alarm" );
}
```

12.3.2 Authorization Example

This section demonstrates a possible use of the User Admin service. The service has a flexible model and many other schemes are possible.

Assume an Operator installs an OSGi Service Platform. Bundles in this environment have defined the following action groups:

```
AlarmSystemControl
InternetAccess
TemperatureControl
PhotoAlbumEdit
PhotoAlbumView
PortForwarding
```

Installing and uninstalling bundles could potentially extend this set. Therefore, the Operator also defines a number of groups that can be used to contain the different types of system users.

```
Administrators
Buddies
Children
Adults
Residents
```

In a particular instance, the Operator installs it in a household with the following residents and buddies:

```
Residents:          Elmer, Fudd, Marvin, Pepe
Buddies:            Daffy, Foghorn
```

First, the residents and buddies are assigned to the system user groups. Second, the user groups need to be assigned to the action groups.

The following tables show how the groups could be assigned.

Groups	Elmer	Fudd	Marvin	Pepe	Daffy	Foghorn
Residents	Basic	Basic	Basic	Basic	-	-
Buddies	-	-	-	-	Basic	Basic
Children	-	-	Basic	Basic	-	-
Adults	Basic	Basic	-	-	-	-
Administrators	Basic	-	-	-	-	-

Table 16 *Example Groups with Basic and Required Members*

Groups	Residents	Buddies	Children	Adults	Admin
AlarmSystemControl	Basic	-	-	-	Required
InternetAccess	Basic	-	-	Required	-
TemperatureControl	Basic	-	-	Required	-
PhotoAlbumEdit	Basic	-	Basic	Basic	-
PhotoAlbumView	Basic	Basic	-	-	-
PortForwarding	Basic	-	-	-	Required

Table 17 *Example Action Groups with their Basic and Required Members*

12.4 Repository Maintenance

The UserAdmin interface is a straightforward API to maintain a repository of User and Group objects. It contains methods to create new Group and User objects with the createRole(String,int) method. The method is prepared so that the same signature can be used to create new types of roles in the future. The interface also contains a method to remove a Role object.

The existing configuration can be obtained with methods that list all Role objects using a filter argument. This filter, which has the same syntax as the Framework filter, must only return the Role objects for which the filter matches the properties.

Several utility methods simplify getting User objects depending on their properties.

12.5 User Admin Events

Changes in the User Admin service can be determined in real time. Each User Admin service implementation must send a UserAdminEvent object to any service in the Framework service registry that is registered under the UserAdminListener interface.

This procedure is demonstrated in the following code sample.

```
class Listener implements UserAdminListener {
   public void roleChanged( UserAdminEvent event ) {
      ...
   }
}
public class MyActivator
   implements BundleActivator {
   public void start( BundleContext context ) {
      context.registerService(
         UserAdminListener.class.getName(),
         new Listener(), null );
   }
   public void stop( BundleContext context ) {}
}
```

It is not necessary to unregister the listener object when the bundle is stopped because the Framework automatically unregisters it. Once registered, the UserAdminListener object must be notified of all changes to the role repository.

12.6 Security

The User Admin service is related to the security model of the OSGi Service Platform, but is complementary to the [33] *The Java Security Architecture for JDK 1.2*. The final permission of most code should be the intersection of the Java 2 Permissions, which are based on the code that is executing, and the User Admin service authorization, which is based on the user for whom the code runs.

12.6.1 UserAdminPermission

The User Admin service defines the UserAdminPermission class that can be used to restrict bundles in accessing credentials. This permission class has the following actions:

- changeProperty – This permission is required to modify properties. The name of the permission is the prefix of the property name.
- changeCredential – This action permits changing credentials. The name of the permission is the prefix of the name of the credential.
- getCredential – This action permits getting credentials. The name of the permission is the prefix of the credential.

If the name of the permission is "admin", it allows the owner to administer the repository. No action is associated with the permission in that case.

Otherwise, the permission name is used to match the property name. This name may end with a ".*" string to indicate a wildcard. For example, com.acme.*matches com.acme.fudd.elmer and com.acme.bugs.

12.7 Relation to JAAS

At a glance, the Java Authorization and Authentication Service (JAAS) seems to be a very suitable model for user administration. The OSGi organization, however, decided to develop an independent User Admin service because JAAS was not deemed applicable. The reasons for this include dependency on J2SE version 1.3 ("JDK 1.3") and existing mechanisms in the previous OSGi Service Gateway 1.0 specification.

12.7.1 JDK 1.3 Dependencies

The authorization component of JAAS relies on the java.security.DomainCombiner interface, which provides a means to dynamically update the ProtectionDomain objects affiliated with an AccessControlContext object.

This interface was added in JDK 1.3. In the context of JAAS, the SubjectDomainCombiner object, which implements the DomainCombiner interface, is used to update ProtectionDomain objects. The permissions of ProtectionDomain objects depend on where code came from and who signed it, with permissions based on who is running the code.

Leveraging JAAS would have resulted in user-based access control on the OSGi Service Platform being available only with JDK 1.3, which was not deemed acceptable.

12.7.2 Existing OSGi Mechanism

JAAS provides a pluggable authentication architecture, which enables applications and their underlying authentication services to remain independent from each other.

The Http Service already provides a similar feature by allowing servlet and resource registrations to be supported by an HttpContext object, which uses a callback mechanism to perform any required authentication checks before granting access to the servlet or resource. This way, the registering bundle has complete control on a per-servlet and per-resource basis over which authentication protocol to use, how the credentials presented by the remote requestor are to be validated, and who should be granted access to the servlet or resource.

12.7.3 Future Road Map

In the future, the main barrier of 1.3 compatibility will be removed. JAAS could then be implemented in an OSGi environment. At that time, the User Admin service will still be needed and will provide complementary services in the following ways:

- The authorization component relies on group membership information to be stored and managed outside JAAS. JAAS does not manage persistent information, so the User Admin service can be a provider of group information when principals are assigned to a Subject object.
- The authorization component allows for credentials to be collected and verified, but a repository is needed to actually validate the credentials.

In the future, the User Admin service can act as the back-end database to JAAS. The only aspect JAAS will remove from the User Admin service is the need for the Authorization interface.

12.8 Changes

The description of the Http Service authentication has been removed because it duplicated the description in the Http Service Specification.

12.9 org.osgi.service.useradmin

The OSGi User Admin service 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.useradmin; specification-version=1.0
```

12.9.1 Summary

- *Authorization* – The Authorization interface encapsulates an authorization context on which bundles can base authorization decisions, where appropriate. [p.264]
- *Group* – A named grouping of roles (Role objects). [p.265]

- *Role* – The base interface for Role objects managed by the User Admin service. [p.267]
- *User* – A User role managed by a User Admin service. [p.268]
- *UserAdmin* – This interface is used to manage a database of named Role objects, which can be used for authentication and authorization purposes. [p.269]
- *UserAdminEvent* – Role change event. [p.271]
- *UserAdminListener* – Listener for UserAdminEvents. [p.272]
- *UserAdminPermission* – Permission to configure and access the Role[p.267] objects managed by a User Admin service. [p.272]

12.9.2 public interface Authorization

The Authorization interface encapsulates an authorization context on which bundles can base authorization decisions, where appropriate.

Bundles associate the privilege to access restricted resources or operations with roles. Before granting access to a restricted resource or operation, a bundle will check if the Authorization object passed to it possess the required role, by calling its hasRole method.

Authorization contexts are instantiated by calling the UserAdmin.getAuthorization[p.270] method.

Trusting Authorization objects

There are no restrictions regarding the creation of Authorization objects. Hence, a service must only accept Authorization objects from bundles that has been authorized to use the service using code based (or Java 2) permissions.

In some cases it is useful to use ServicePermission to do the code based access control. A service basing user access control on Authorization objects passed to it, will then require that a calling bundle has the ServicePermission to get the service in question. This is the most convenient way. The OSGi environment will do the code based permission check when the calling bundle attempts to get the service from the service registry.

Example: A servlet using a service on a user's behalf. The bundle with the servlet must be given the ServicePermission to get the Http Service.

However, in some cases the code based permission checks need to be more fine-grained. A service might allow all bundles to get it, but require certain code based permissions for some of its methods.

Example: A servlet using a service on a user's behalf, where some service functionality is open to anyone, and some is restricted by code based permissions. When a restricted method is called (e.g., one handing over an Authorization object), the service explicitly checks that the calling bundle has permission to make the call.

12.9.2.1 public String getName( )

- Gets the name of the User[p.268] that this Authorization context was created for.

Returns The name of the User[p.268] object that this Authorization context was created for, or null if no user was specified when this Authorization context was created.

12.9.2.2 public String[] getRoles( )

- Gets the names of all roles encapsulated by this Authorization context.

Returns The names of all roles encapsulated by this Authorization context, or null if no roles are in the context. The predefined role user.anyone will not be included in this list.

12.9.2.3 public boolean hasRole(String name)

name The name of the role to check for.

- Checks if the role with the specified name is implied by this Authorization context.

  Bundles must define globally unique role names that are associated with the privilege of accessing restricted resources or operations. Operators will grant users access to these resources, by creating a Group[p.265] object for each role and adding User[p.268] objects to it.

Returns true if this Authorization context implies the specified role, otherwise false.

12.9.3 public interface Group extends User

A named grouping of roles (Role objects).

Whether or not a given Authorization context implies a Group object depends on the members of that Group object.

A Group object can have two kinds of members: *basic* and *required*. A Group object is implied by an Authorization context if all of its required members are implied and at least one of its basic members is implied.

A Group object must contain at least one basic member in order to be implied. In other words, a Group object without any basic member roles is never implied by any Authorization context.

A User object always implies itself.

No loop detection is performed when adding members to Group objects, which means that it is possible to create circular implications. Loop detection is instead done when roles are checked. The semantics is that if a role depends on itself (i.e., there is an implication loop), the role is not implied.

The rule that a Group object must have at least one basic member to be implied is motivated by the following example:

```
group foo
  required members: marketing
  basic members: alice, bob
```

Privileged operations that require membership in “foo” can be performed only by “alice” and “bob”, who are in marketing.

If "alice" and "bob" ever transfer to a different department, anybody in marketing will be able to assume the "foo" role, which certainly must be prevented. Requiring that "foo" (or any Group object for that matter) must have at least one basic member accomplishes that.

However, this would make it impossible for a Group object to be implied by just its required members. An example where this implication might be useful is the following declaration: "Any citizen who is an adult is allowed to vote." An intuitive configuration of "voter" would be:

```
group voter
  required members: citizen, adult
     basic members:
```

However, according to the above rule, the "voter" role could never be assumed by anybody, since it lacks any basic members. In order to address this issue a predefined role named "user.anyone" can be specified, which is always implied. The desired implication of the "voter" group can then be achieved by specifying "user.anyone" as its basic member, as follows:

```
group voter
  required members: citizen, adult
     basic members: user.anyone
```

12.9.3.1 public boolean addMember(Role role)

role The role to add as a basic member.

□ Adds the specified Role object as a basic member to this Group object.

Returns true if the given role could be added as a basic member, and false if this Group object already contains a Role object whose name matches that of the specified role.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.

12.9.3.2 public boolean addRequiredMember(Role role)

role The Role object to add as a required member.

□ Adds the specified Role object as a required member to this Group object.

Returns true if the given Role object could be added as a required member, and false if this Group object already contains a Role object whose name matches that of the specified role.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.

12.9.3.3 public Role[] getMembers( )

□ Gets the basic members of this Group object.

Returns The basic members of this Group object, or null if this Group object does not contain any basic members.

12.9.3.4 public Role[] getRequiredMembers( )

□ Gets the required members of this Group object.

Returns The required members of this Group object, or null if this Group object does not contain any required members.

12.9.3.5 public boolean removeMember(Role role)

role The Role object to remove from this Group object.

□ Removes the specified Role object from this Group object.

Returns true if the Role object could be removed, otherwise false.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.

12.9.4 public interface Role

The base interface for Role objects managed by the User Admin service.

This interface exposes the characteristics shared by all Role classes: a name, a type, and a set of properties.

Properties represent public information about the Role object that can be read by anyone. Specific UserAdminPermission[p.272] objects are required to change a Role object's properties.

Role object properties are Dictionary objects. Changes to these objects are propagated to the User Admin service and made persistent.

Every User Admin service contains a set of predefined Role objects that are always present and cannot be removed. All predefined Role objects are of type ROLE. This version of the org.osgi.service.useradmin package defines a single predefined role named "user.anyone", which is inherited by any other role. Other predefined roles may be added in the future. Since "user.anyone" is a Role object that has properties associated with it that can be read and modified. Access to these properties and their use is application specific and is controlled using UserAdminPermission in the same way that properties for other Role objects are.

12.9.4.1 public static final int GROUP = 2

The type of a Group[p.265] role.

The value of GROUP is 2.

12.9.4.2 public static final int ROLE = 0

The type of a predefined role.

The value of ROLE is 0.

12.9.4.3 public static final int USER = 1

The type of a User[p.268] role.

The value of USER is 1.

12.9.4.4 public String getName( )

□ Returns the name of this role.

Returns The role's name.

12.9.4.5 public Dictionary getProperties( )

- Returns a Dictionary of the (public) properties of this Role object. Any changes to the returned Dictionary will change the properties of this Role object. This will cause a UserAdminEvent object of type UserAdminEvent.ROLE_CHANGED[p.271] to be broadcast to any UserAdminListener objects.

  Only objects of type String may be used as property keys, and only objects of type String or byte[] may be used as property values. Any other types will cause an exception of type IllegalArgumentException to be raised.

  In order to add, change, or remove a property in the returned Dictionary, a UserAdminPermission[p.272] named after the property name (or a prefix of it) with action changeProperty is required.

Returns Dictionary containing the properties of this Role object.

12.9.4.6 public int getType( )

- Returns the type of this role.

Returns The role's type.

12.9.5 public interface User extends Role

A User role managed by a User Admin service.

In this context, the term "user" is not limited to just human beings. Instead, it refers to any entity that may have any number of credentials associated with it that it may use to authenticate itself.

In general, User objects are associated with a specific User Admin service (namely the one that created them), and cannot be used with other User Admin services.

A User object may have credentials (and properties, inherited from the Role[p.267] class) associated with it. Specific UserAdminPermission[p.272] objects are required to read or change a User object's credentials.

Credentials are Dictionary objects and have semantics that are similar to the properties in the Role class.

12.9.5.1 public Dictionary getCredentials( )

- Returns a Dictionary of the credentials of this User object. Any changes to the returned Dictionary object will change the credentials of this User object. This will cause a UserAdminEvent object of type UserAdminEvent.ROLE_CHANGED[p.271] to be broadcast to any UserAdminListeners objects.

  Only objects of type String may be used as credential keys, and only objects of type String or of type byte[] may be used as credential values. Any other types will cause an exception of type IllegalArgumentException to be raised.

  In order to retrieve a credential from the returned Dictionary object, a UserAdminPermission[p.272] named after the credential name (or a prefix of it) with action getCredential is required.

In order to add or remove a credential from the returned Dictionary object, a UserAdminPermission[p.272] named after the credential name (or a prefix of it) with action changeCredential is required.

Returns Dictionary object containing the credentials of this User object.

12.9.5.2 public boolean hasCredential(String key, Object value)

key The credential key.

value The credential value.

- Checks to see if this User object has a credential with the specified key set to the specified value.

  If the specified credential value is not of type String or byte[], it is ignored, that is, false is returned (as opposed to an IllegalArgumentException being raised).

Returns true if this user has the specified credential; false otherwise.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission named after the credential key (or a prefix of it) with action getCredential.

12.9.6 public interface UserAdmin

This interface is used to manage a database of named Role objects, which can be used for authentication and authorization purposes.

This version of the User Admin service defines two types of Role objects: "User" and "Group". Each type of role is represented by an int constant and an interface. The range of positive integers is reserved for new types of roles that may be added in the future. When defining proprietary role types, negative constant values must be used.

Every role has a name and a type.

A User[p.268] object can be configured with credentials (e.g., a password) and properties (e.g., a street address, phone number, etc.).

A Group[p.265] object represents an aggregation of User[p.268] and Group[p.265] objects. In other words, the members of a Group object are roles themselves.

Every User Admin service manages and maintains its own namespace of Role objects, in which each Role object has a unique name.

12.9.6.1 public Role createRole(String name, int type)

name The name of the Role object to create.

type The type of the Role object to create. Must be either a Role.USER[p.267] type or Role.GROUP[p.267] type.

- Creates a Role object with the given name and of the given type.

  If a Role object was created, a UserAdminEvent object of type UserAdminEvent.ROLE_CREATED[p.271] is broadcast to any UserAdminListener object.

Returns The newly created Role object, or null if a role with the given name already exists.

Throws IllegalArgumentException – if type is invalid.

SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.

12.9.6.2 public Authorization getAuthorization(User user)

user The User object to create an Authorization object for, or null for the anonymous user.

- Creates an Authorization object that encapsulates the specified User object and the Role objects it possesses. The null user is interpreted as the anonymous user. The anonymous user represents a user that has not been authenticated. An Authorization object for an anonymous user will be unnamed, and will only imply groups that user.anyone implies.

Returns the Authorization object for the specified User object.

12.9.6.3 public Role getRole(String name)

name The name of the Role object to get.

- Gets the Role object with the given name from this User Admin service.

Returns The requested Role object, or null if this User Admin service does not have a Role object with the given name.

12.9.6.4 public Role[] getRoles(String filter) throws InvalidSyntaxException

filter The filter criteria to match.

- Gets the Role objects managed by this User Admin service that have properties matching the specified LDAP filter criteria. See org.osgi.framework.Filter for a description of the filter syntax. If a null filter is specified, all Role objects managed by this User Admin service are returned.

Returns The Role objects managed by this User Admin service whose properties match the specified filter criteria, or all Role objects if a null filter is specified. If no roles match the filter, null will be returned.

12.9.6.5 public User getUser(String key, String value)

key The property key to look for.

value The property value to compare with.

- Gets the user with the given property key-value pair from the User Admin service database. This is a convenience method for retrieving a User object based on a property for which every User object is supposed to have a unique value (within the scope of this User Admin service), such as for example a X.500 distinguished name.

Returns A matching user, if exactly one is found. If zero or more than one matching users are found, null is returned.

12.9.6.6 public boolean removeRole(String name)

name The name of the Role object to remove.

- Removes the Role object with the given name from this User Admin service.

If the Role object was removed, a UserAdminEvent object of type UserAdminEvent.ROLE_REMOVED[p.271] is broadcast to any UserAdminListener object.

Returns true If a Role object with the given name is present in this User Admin service and could be removed, otherwise false.

Throws SecurityException – If a security manager exists and the caller does not have the UserAdminPermission with name admin.

12.9.7 public class UserAdminEvent

Role change event.

UserAdminEvent objects are delivered asynchronously to any UserAdminListener objects when a change occurs in any of the Role objects managed by a User Admin service.

A type code is used to identify the event. The following event types are defined: ROLE_CREATED[p.271] type, ROLE_CHANGED[p.271] type, and ROLE_REMOVED[p.271] type. Additional event types may be defined in the future.

See Also UserAdmin[p.269] , UserAdminListener[p.272]

12.9.7.1 public static final int ROLE_CHANGED = 2

A Role object has been modified.

The value of ROLE_CHANGED is 0x00000002.

12.9.7.2 public static final int ROLE_CREATED = 1

A Role object has been created.

The value of ROLE_CREATED is 0x00000001.

12.9.7.3 public static final int ROLE_REMOVED = 4

A Role object has been removed.

The value of ROLE_REMOVED is 0x00000004.

12.9.7.4 public UserAdminEvent(ServiceReference ref, int type, Role role)

ref The ServiceReference object of the User Admin service that generated this event.

type The event type.

role The Role object on which this event occurred.

- Constructs a UserAdminEvent object from the given ServiceReference object, event type, and Role object.

12.9.7.5 public Role getRole( )

- Gets the Role object this event was generated for.

Returns The Role object this event was generated for.

12.9.7.6 public ServiceReference getServiceReference( )

- Gets the ServiceReference object of the User Admin service that generated this event.

Returns The User Admin service's ServiceReference object.

12.9.7.7 public int getType( )

- Returns the type of this event.

  The type values are ROLE_CREATED[p.271] type, ROLE_CHANGED[p.271] type, and ROLE_REMOVED[p.271] type.

Returns The event type.

12.9.8 public interface UserAdminListener

Listener for UserAdminEvents.

UserAdminListener objects are registered with the Framework service registry and notified with a UserAdminEvent object when a Role object has been created, removed, or modified.

UserAdminListener objects can further inspect the received UserAdminEvent object to determine its type, the Role object it occurred on, and the User Admin service that generated it.

See Also UserAdmin[p.269] , UserAdminEvent[p.271]

12.9.8.1 public void roleChanged(UserAdminEvent event)

event The UserAdminEvent object.

- Receives notification that a Role object has been created, removed, or modified.

12.9.9 public final class UserAdminPermission extends BasicPermission

Permission to configure and access the Role[p.267] objects managed by a User Admin service.

This class represents access to the Role objects managed by a User Admin service and their properties and credentials (in the case of User[p.268] objects).

The permission name is the name (or name prefix) of a property or credential. The naming convention follows the hierarchical property naming convention. Also, an asterisk may appear at the end of the name, following a ".", or by itself, to signify a wildcard match. For example: "org.osgi.security.protocol.*" or "*" is valid, but "*protocol" or "a*b" are not valid.

The UserAdminPermission with the reserved name "admin" represents the permission required for creating and removing Role objects in the User Admin service, as well as adding and removing members in a Group object. This UserAdminPermission does not have any actions associated with it.

The actions to be granted are passed to the constructor in a string containing a list of one or more comma-separated keywords. The possible keywords are: changeProperty, changeCredential, and getCredential. Their meaning is defined as follows:

```
 action
 changeProperty    Permission to change (i.e., add and
remove)
                   Role object properties whose names start
with
                   the name argument specified in the con-
structor.
 changeCredential  Permission to change (i.e., add and
remove)
                   User object credentials whose names start
                   with the name argument specified in the
constructor.
 getCredential     Permission to retrieve and check for the
                   existence of User object credentials whose
names
                   start with the name argument specified in
the
                   constructor.
```

The action string is converted to lowercase before processing.

Following is a PermissionInfo style policy entry which grants a user administration bundle a number of UserAdminPermission object:

```
 (org.osgi.service.useradmin.UserAdminPermission "admin")
 (org.osgi.service.useradmin.UserAdminPermission "com.foo.*"
"changeProperty,getCredential,changeCredential")
 (org.osgi.service.useradmin.UserAdminPermission "user.*",
"changeProperty,changeCredential")
```

The first permission statement grants the bundle the permission to perform any User Admin service operations of type "admin", that is, create and remove roles and configure Group objects.

The second permission statement grants the bundle the permission to change any properties as well as get and change any credentials whose names start with com.foo..

The third permission statement grants the bundle the permission to change any properties and credentials whose names start with user.. This means that the bundle is allowed to change, but not retrieve any credentials with the given prefix.

The following policy entry empowers the Http Service bundle to perform user authentication:

```
grant codeBase "${jars}http.jar" {
  permission org.osgi.service.useradmin.UserAdminPermission
    "user.password", "getCredential";
};
```

The permission statement grants the Http Service bundle the permission to validate any password credentials (for authentication purposes), but the bundle is not allowed to change any properties or credentials.

12.9.9.1 public static final String ADMIN = "admin"

The permission name "admin".

12.9.9.2 public static final String CHANGE_CREDENTIAL = "changeCredential"

The action string "changeCredential".

12.9.9.3 public static final String CHANGE_PROPERTY = "changeProperty"

The action string "changeProperty".

12.9.9.4 public static final String GET_CREDENTIAL = "getCredential"

The action string "getCredential".

12.9.9.5 public UserAdminPermission(String name, String actions)

name the name of this `UserAdminPermission`

actions the action string.

- Creates a new `UserAdminPermission` with the specified name and actions. `name` is either the reserved string "admin" or the name of a credential or property, and `actions` contains a comma-separated list of the actions granted on the specified name. Valid actions are `changeProperty`, `changeCredential`, and getCredential.

Throws `IllegalArgumentException` – If `name` equals "admin" and `actions` are specified.

12.9.9.6 public boolean equals(Object obj)

obj the object to be compared for equality with this object.

- Checks two `UserAdminPermission` objects for equality. Checks that `obj` is a `UserAdminPermission`, and has the same name and actions as this object.

Returns `true` if `obj` is a `UserAdminPermission` object, and has the same name and actions as this `UserAdminPermission` object.

12.9.9.7 public String getActions( )

- Returns the canonical string representation of the actions, separated by comma.

Returns the canonical string representation of the actions.

12.9.9.8 public int hashCode( )

- Returns the hash code of this `UserAdminPermission` object.

12.9.9.9 public boolean implies(Permission p)

p the permission to check against.

- Checks if this `UserAdminPermission` object "implies" the specified permission.

  More specifically, this method returns `true` if:

- *p* is an instanceof UserAdminPermission,
- *p*'s actions are a proper subset of this object's actions, and
- *p*'s name is implied by this object's name. For example, "java.*" implies "java.home".

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

12.9.9.10 public PermissionCollection newPermissionCollection( )

☐ Returns a new PermissionCollection object for storing UserAdminPermission objects.

Returns a new PermissionCollection object suitable for storing UserAdminPermission objects.

12.9.9.11 public String toString( )

☐ Returns a string describing this UserAdminPermission object. This string must be in PermissionInfo encoded format.

Returns The PermissionInfo encoded string for this UserAdminPermission object.

See Also org.osgi.service.permissionadmin.PermissionInfo.getEncoded

12.10 References

[33] *The Java Security Architecture for JDK 1.2*
Version 1.0, Sun Microsystems, October 1998
http://java.sun.com/products/jdk/1.4/docs/guide/security/spec/security-spec.doc.html

[34] *Java Authentication and Authorization Service*
http://java.sun.com/products/jaas

13 IO Connector Service Specification

Version 1.0

13.1 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 [35] *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.

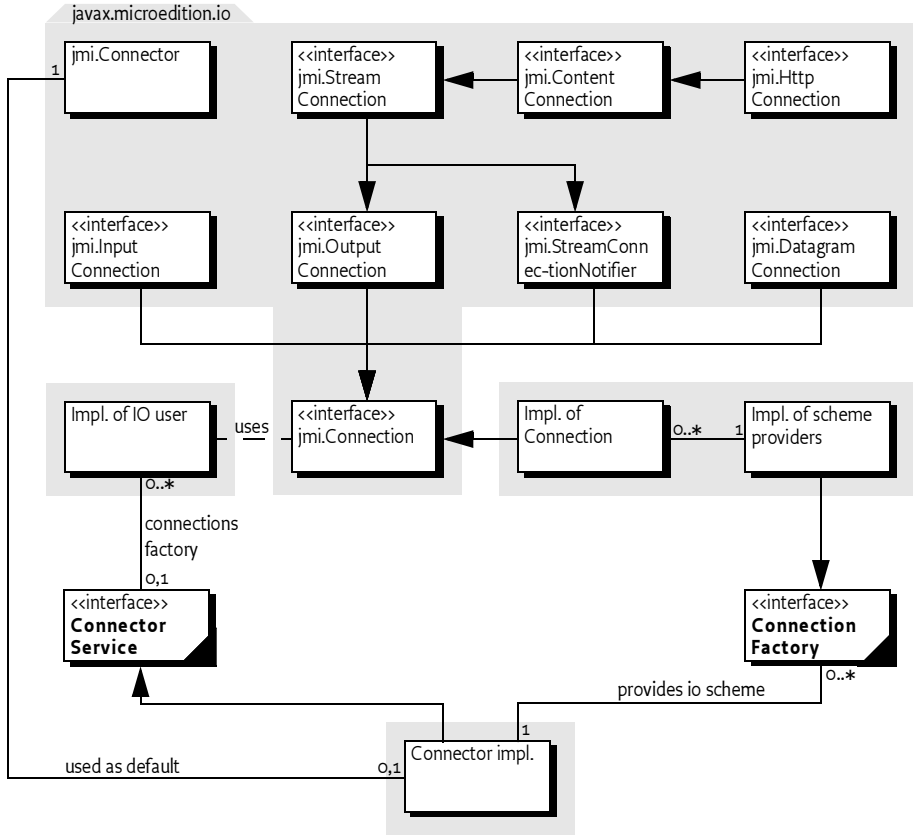
13.1.1 Essentials

- *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.

13.1.2 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 46 Class Diagram, org.osgi.service.io (jmi is javax.microedition.io)



13.2 The Connector Framework

The [35] *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 ) { ... }
      }
   }
```

13.3 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.

13.4 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() {}
}
```

13.4.1 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.

13.5 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.

13.6 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.

13.7 org.osgi.service.io

The OSGi IO Connector Specification Version 1.0.

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

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

13.7.1 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.281]
- ConnectorService – The Connector Service should be called to create and open javax.microedition.io.Connection objects. [p.283]

13.7.2 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.283] method of the selected Connection Factory will then be called to create the actual Connection object.

13.7.2.1 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[].

13.7.2.2 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.

13.7.3 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`.

13.7.3.1 public static final int READ = 1

Read access mode.

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

13.7.3.2 public static final int READ_WRITE = 3

Read/Write access mode.

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

13.7.3.3 public static final int WRITE = 2

Write access mode.

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

13.7.3.4 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)`

13.7.3.5 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)

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

name The URI for the connection.

mode The access mode.

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

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

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

Throws IllegalArgumentException – If a parameter is invalid.

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

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

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

13.7.3.7 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)

13.7.3.8 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)`

13.7.3.9 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)`

13.7.3.10 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)`

13.8 References

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

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

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

14 Http Service Specification

Version 1.1

14.1 Introduction

An OSGi Service Platform normally provides users with access to services on the Internet and other networks. This access allows users to remotely retrieve information from, and send control to, services in an OSGi Service Platform using a standard web browser.

Bundle developers typically need to develop communication and user interface solutions for standard technologies such as HTTP, HTML, XML, and servlets.

The Http Service supports two standard techniques for this purpose:

- *Registering servlets* – A servlet is a Java object which implements the Java Servlet API. Registering a servlet in the Framework gives it control over some part of the Http Service URI name-space.
- *Registering resources* – Registering a resource allows HTML files, image files, and other static resources to be made visible in the Http Service URI name-space by the requesting bundle.

Implementations of the Http Service can be based on:

- [38] *HTTP 1.0 Specification RFC-1945*
- [39] *HTTP 1.1 Specification RFC-2616*

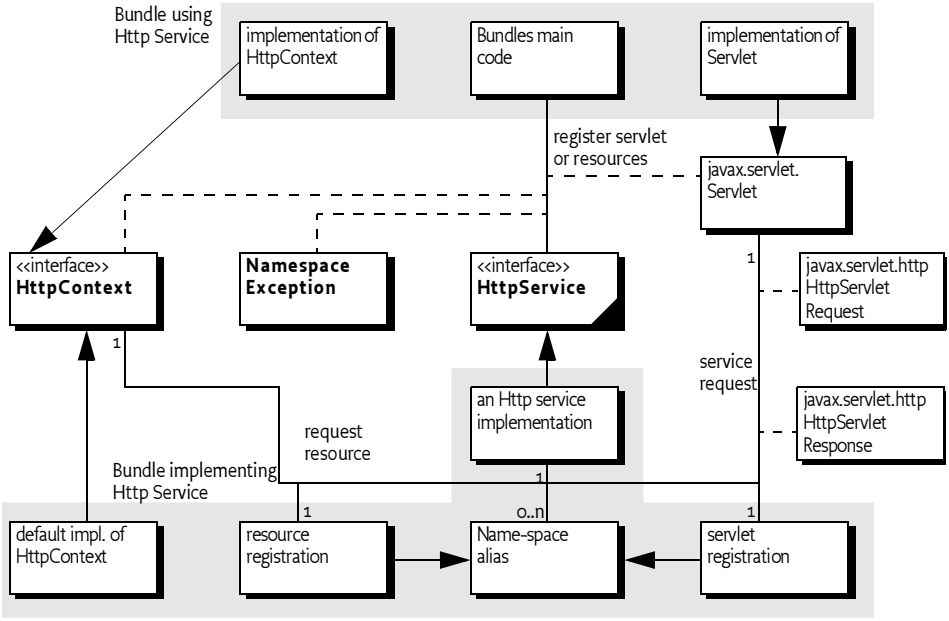
Alternatively, implementations of this service can support other protocols if these protocols can conform to the semantics of the javax.servlet API. This additional support is necessary because the Http Service is closely related to [40] *Java Servlet Technology*. Http Service implementations must support at least version 2.1 of the Java Servlet API.

14.1.1 Entities

This specification defines the following interfaces which a bundle developer can implement collectively as an Http Service or use individually:

- HttpContext – Allows bundles to provide information for a servlet or resource registration.
- HttpService – Allows other bundles in the Framework to dynamically register and unregister resources and servlets into the Http Service URI name-space.
- NamespaceException – Is thrown to indicate an error with the caller's request to register a servlet or resource into the Http Service URI name-space.

Figure 47 Http Service Overview Diagram



14.2 Registering Servlets

javax.servlet.Servlet objects can be registered with the Http Service by using the HttpService interface. For this purpose, the HttpService interface defines the method registerServlet(String,javax.servlet.Servlet,Dictionary,HttpContext).

For example, if the Http Service implementation is listening to port 80 on the machine www.acme.com and the Servlet object is registered with the name "/servlet", then the Servlet object's service method is called when the following URL is used from a web browser:

```
http://www.acme.com/servlet?name=bugs
```

All Servlet objects and resource registrations share the same name-space. If an attempt is made to register a resource or Servlet object under the same name as a currently registered resource or Servlet object, a NamespaceException is thrown. See *Mapping HTTP Requests to Servlet and Resource Registrations* on page 292 for more information about the handling of the Http Service name-space.

Each Servlet registration must be accompanied with an HttpContext object. This object provides the handling of resources, media typing, and a method to handle authentication of remote requests. See *Authentication* on page 295.

For convenience, a default HttpContext object is provided by the Http Service and can be obtained with createDefaultHttpContext(). Passing a null parameter to the registration method achieves the same effect.

Servlet objects require a ServletContext object. This object provides a number of functions to access the Http Service Java Servlet environment. It is created by the implementation of the Http Service for each unique HttpContext object with which a Servlet object is registered. Thus, Servlet objects registered with the same HttpContext object must also share the same ServletContext object.

Servlet objects are initialized by the Http Service when they are registered and bound to that specific Http Service. The initialization is done by calling the Servlet object's Servlet.init(ServletConfig) method. The ServletConfig parameter provides access to the initialization parameters specified when the Servlet object was registered.

Therefore, the same Servlet instance must not be reused for registration with another Http Service, nor can it be registered under multiple names. Unique instances are required for each registration.

The following example code demonstrates the use of the registerServlet method:

```
Hashtable initparams = new Hashtable();
initparams.put( "name", "value" );

Servlet myServlet = new HttpServlet() {
   String     name = "<not set>";

   public void init( ServletConfig config ) {
      this.name = (String)
         config.getInitParameter( "name" );
   }

   public void doGet(
      HttpServletRequest req,
      HttpServletResponse rsp
   ) throws IOException {
      rsp.setContentType( "text/plain" );
      req.getWriter().println( this.name );
   }
};

getHttpService().registerServlet(
   "/servletAlias",
   myServlet,
   initparams,
   null // use default context
);
// myServlet has been registered
// and its init method has been called. Remote
// requests are now handled and forwarded to
// the servlet.
...
getHttpService().unregister("/servletAlias");
// myServlet has been unregistered and its
// destroy method has been called
```

This example registers the servlet, myServlet, at alias: /servletAlias. Future requests for http://www.acme.com/servletAlias maps to the servlet, myServlet, whose service method is called to process the request. (The service method is called in the HttpServlet base class and dispatched to a doGet, doPut, doPost, doOptions, doTrace, or doDelete call depending on the HTTP request method used.)

14.3 Registering Resources

A resource is a file containing images, static HTML pages, sounds, movies, applets, etc. Resources do not require any handling from the bundle. They are transferred directly from their source–usually the JAR file that contains the code for the bundle–to the requestor using HTTP.

Resources could be handled by Servlet objects as explained in *Registering Servlets* on page 288. Transferring a resource over HTTP, however, would require very similar Servlet objects for each bundle. To prevent this redundancy, resources can be registered directly with the Http Service via the HttpService interface. This HttpService interface defines the registerResources(String,String,HttpContext)method for registering a resource into the Http Service URI name-space.

The first parameter is the external alias under which the resource is registered with the Http Service. The second parameter is an internal prefix to map this resource to the bundle's name-space. When a request is received, the HttpService object must remove the external alias from the URI, replace it with the internal prefix, and call the getResource(String) method with this new name on the associated HttpContext object. The HttpContext object is further used to get the MIME type of the resource and to authenticate the request.

Resources are returned as a java.net.URL object. The Http Service must read from this URL object and transfer the content to the initiator of the HTTP request.

This return type was chosen because it matches the return type of the java.lang.Class.getResource(String resource) method. This method can retrieve resources directly from the same place as the one from which the class was loaded – often a package directory in the JAR file of the bundle. This method makes it very convenient to retrieve resources from the bundle that are contained in the package.

The following example code demonstrates the use of the registerResources method:

```
package com.acme;
...
HttpContext context = new HttpContext() {
   public boolean handleSecurity(
      HttpServletRequest request,
      HttpServletResponse response
   ) throws IOException {
      return true;
```

```
    }

    public URL getResource(String name) {
       return getClass().getResource(name);
    }

    public String getMimeType(String name) {
       return null;
    }
};

getHttpService().registerResources (
    "/files",
    "www",
    context
);
...
getHttpService().unregister("/files");
```

This example registers the alias /files on the Http Service. Requests for resources below this name-space are transferred to the HttpContext object with an internal name of www/<name>. This example uses the Class.get Resource(String) method. Because the internal name does not start with a "/", it must map to a resource in the "com/acme/www" directory of the JAR file. If the internal name did start with a "/", the package name would not have to be prefixed and the JAR file would be searched from the root. Consult the java.lang.Class.getResource(String) method for more information.

In the example, a request for http://www.acme.com/files/myfile.html must map to the name "com/acme/www/myfile.html" which is in the bundle's JAR file.

More sophisticated implementations of the getResource(String) method could filter the input name, restricting the resources that may be returned or map the input name onto the file system (if the security implications of this action are acceptable).

Alternatively, the resource registration could have used a default HttpContext object, as demonstrated in the following call to registerResources:

```
getHttpService().registerResources(
    "/files",
    "/com/acme/www",
    null
);
```

In this case, the Http Service implementation would call the createDefaultHttpContext() method and use its return value as the HttpContext argument for the registerResources method. The default implementation must map the resource request to the bundle's resource, using

Bundle.getResource(String). In the case of the previous example, however, the internal name must now specify the full path to the directory containing the resource files in the JAR file. No automatic prefixing of the package name is done.

The getMime(String) implementation of the default HttpContext object should return a reasonable mapping. Its handleSecurity(HttpServlet Request,HttpServletResponse) may implement an authentication mechanism that is implementation-dependent.

14.4 Mapping HTTP Requests to Servlet and Resource Registrations

When an HTTP request comes in from a client, the Http Service checks to see if the requested URI matches any registered aliases. A URI matches only if the path part of the URI is exactly the same string. Matching is case sensitive.

If it does match, a matching registration takes place, which is processed as follows:

1. If the registration corresponds to a servlet, the authorization is verified by calling the handleSecurity method of the associated HttpContext object. See *Authentication* on page 295. If the request is authorized, the servlet must be called by its service method to complete the HTTP request.
2. If the registration corresponds to a resource, the authorization is verified by calling the handleSecurity method of the associated HttpContext object. See *Authentication* on page 295. If the request is authorized, a target resource name is constructed from the requested URI by substituting the alias from the registration with the internal name from the registration if the alias is not "/". If the alias is "/", then the target resource name is constructed by prefixing the requested URI with the internal name. An internal name of "/" is considered to have the value of the empty string ("") during this process.
3. The target resource name must be passed to the getResource method of the associated HttpContext object.
4. If the returned URL object is not null, the Http Service must return the contents of the URL to the client completing the HTTP request. The translated target name, as opposed to the original requested URI, must also be used as the argument to HttpContext.getMimeType.
5. If the returned URL object is null, the Http Service continues as if there was no match.
6. If there is no match, the Http Service must attempt to match sub-strings of the requested URI to registered aliases. The sub-strings of the requested URI are selected by removing the last "/" and everything to the right of the last "/".

The Http Service must repeat this process until either a match is found or the sub-string is an empty string. If the sub-string is empty and the alias "/" is registered, the request is considered to match the alias "/". Otherwise, the Http Service must return HttpServletResponse.SC_NOT_FOUND(404) to the client.

For example, an HTTP request comes in with a request URI of "/fudd/bugs/foo.txt", and the only registered alias is "/fudd". A search for "/fudd/bugs/foo.txt" will not match an alias. Therefore, the Http Service will search for the alias "/fudd/bugs" and the alias "/fudd". The latter search will result in a match and the matched alias registration must be used.

Registrations for identical aliases are not allowed. If a bundle registers the alias "/fudd", and another bundle tries to register the exactly the same alias, the second caller must receive a NamespaceException and its resource or servlet must *not* be registered. It could, however, register a similar alias – for example, "/fudd/bugs", as long as no other registration for this alias already exists.

The following table shows some examples of the usage of the name-space.

Alias	Internal Name	URI	getResource Parameter
/	(empty string)	/fudd/bugs	/fudd/bugs
/	/	/fudd/bugs	/fudd/bugs
/	/tmp	/fudd/bugs	/tmp/bugs
/fudd	(empty string)	/fudd/bugs	/bugs
/fudd	/	/fudd/bugs	/bugs
/fudd	/tmp	/fudd/bugs	/tmp/bugs
/fudd	tmp	/fudd/bugs/x.gif	tmp/bugs/x.gif
/fudd/bugs/x.gif	tmp/y.gif	/fudd/bugs/x.gif	tmp/y.gif

Table 18 *Examples of Name-space Mapping*

14.5 The Default Http Context Object

The HttpContext object in the first example demonstrates simple implementations of the HttpContext interface methods. Alternatively, the example could have used a default HttpContext object, as demonstrated in the following call to registerServlet:

```
getHttpService().registerServlet(
   "/servletAlias",
   myServlet,
   initparams,
   null
);
```

In this case, the Http Service implementation must call createDefault HttpContext and use the return value as the HttpContext argument.

If the default HttpContext object, and thus the ServletContext object, is to be shared by multiple servlet registrations, the previous servlet registration example code needs to be changed to use the same default HttpContext object. This change is demonstrated in the next example:

```
HttpContext defaultContext =
   getHttpService().createDefaultHttpContext();

getHttpService().registerServlet(
   "/servletAlias",
   myServlet,
   initparams,
   defaultContext
);

// defaultContext can be reused
// for further servlet registrations
```

14.6 Multipurpose Internet Mail Extension (MIME) Types

MIME defines an extensive set of headers and procedures to encode binary messages in US-ASCII mails. For an overview of all the related RFCs, consult [41] *MIME Multipurpose Internet Mail Extension.*

An important aspect of this extension is the type (file format) mechanism of the binary messages. The type is defined by a string containing a general category (text, application, image, audio and video, multipart, and message) followed by a "/" and a specific media type, as in the example, "text/html" for HTML formatted text files. A MIME type string can be followed by additional specifiers by separating key=value pairs with a ';'. These specifiers can be used, for example, to define character sets as follows:

```
text/plan ; charset=iso-8859-1
```

The Internet Assigned Number Authority (IANA) maintains a set of defined MIME media types. This list can be found at [42] *Assigned MIME Media Types*. MIME media types are extendable, and when any part of the type starts with the prefix "x-", it is assumed to be vendor-specific and can be used for testing. New types can be registered as described in [43] *Registration Procedures for new MIME media types*.

HTTP bases its media typing on the MIME RFCs. The "Content-Type" header should contain a MIME media type so that the browser can recognize the type and format the content correctly.

The source of the data must define the MIME media type for each transfer. Most operating systems do not support types for files, but use conventions based on file names, such as the last part of the file name after the last ".". This extension is then mapped to a media type.

Implementations of the Http Service should have a reasonable default of mapping common extensions to media types based on file extensions.

Extension	MIME media type	Description
.jpg .jpeg	image/jpeg	JPEG Files
.gif	image/gif	GIF Files
.css	text/css	Cascading Style Sheet Files
.txt	text/plain	Text Files
.wml	text/vnd.wap.wml	Wireless Access Protocol (WAP) Mark Language
.htm .html	text/html	Hyper Text Markup Language
.wbmp	image/vnd.wap.wbmp	Bitmaps for WAP

Table 19 *Sample Extension to MIME Media Mapping*

Only the bundle developer, however, knows exactly which files have what media type. The HttpContext interface can therefore be used to map this knowledge to the media type. The HttpContext class has the following method for this: getMimeType(String).

The implementation of this method should inspect the file name and use its internal knowledge to map this name to a MIME media type.

Simple implementations can extract the extension and look up this extension in a table.

Returning null from this method allows the Http Service implementation to use its default mapping mechanism.

14.7 Authentication

The Http Service has separated the authentication and authorization of a request from the execution of the request. This separation allows bundles to use available Servlet sub-classes while still providing bundle specific authentication and authorization of the requests.

Prior to servicing each incoming request, the Http Service calls the handleSecurity(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) method on the HttpContext object that is associated with the request URI. This method controls whether the request is processed in the normal manner or an authentication error is returned.

If an implementation wants to authenticate the request, it can use the authentication mechanisms of HTTP. See [44] *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*. These mechanisms normally interpret the headers and decide if the user identity is available, and if it is, whether that user has authenticated itself correctly.

There are many different ways of authenticating users, and the handleSecurity method on the HttpContext object can use whatever method it requires. If the method returns true, the request must continue to be processed using the potentially modified HttpServletRequest and HttpServletResponse objects. If the method returns false, the request must *not* be processed.

A common standard for HTTP is the basic authentication scheme that is not secure when used with HTTP. Basic authentication passes the password in base 64 encoded strings that are trivial to decode into clear text. Secure transport protocols like HTTPS use SSL to hide this information. With these protocols basic authentication is secure.

Using basic authentication requires the following steps:

1. If no Authorization header is set in the request, the method should set the WWW-Authenticate header in the response. This header indicates the desired authentication mechanism and the realm. For example, WWW-Authenticate: Basic realm="ACME".
   The header should be set with the response object that is given as a parameter to the handleSecurity method. The handleSecurity method should set the status to HttpServletResponse.SC_UNAUTHORIZED (401) and return false.
2. Secure connections can be verified with the ServletRequest.getScheme() method. This method returns, for example, "https" for an SSL connection; the handleSecurity method can use this and other information to decide if the connection's security level is acceptable. If not, the handleSecurity method should set the status to HttpServletResponse.SC_FORBIDDEN (403) and return false.
3. Next, the request must be authenticated. When basic authentication is used, the Authorization header is available in the request and should be parsed to find the user and password. See [44] *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication* for more information.
   If the user cannot be authenticated, the status of the response object should be set to HttpServletResponse.SC_UNAUTHORIZED (401) and return false.
4. The authentication mechanism that is actually used and the identity of the authenticated user can be of interest to the Servlet object. Therefore, the implementation of the handleSecurity method should set this information in the request object using the ServletRequest.setAttribute method. This specification has defined a number of OSGi-specific attribute names for this purpose:
   - AUTHENTICATION_TYPE - Specifies the scheme used in authentication. A Servlet may retrieve the value of this attribute by calling the HttpServletRequest.getAuthType method. This attribute name is org.osgi.service.http.authentication.type.
   - REMOTE_USER - Specifies the name of the authenticated user. A Servlet may retrieve the value of this attribute by calling the HttpServletRequest.getRemoteUser method. This attribute name is org.osgi.service.http.authentication.remote.user.
   - AUTHORIZATION - If a User Admin service is available in the environment, then the handleSecurity method should set this attribute with the Authorization object obtained from the User Admin service. Such an object encapsulates the authentication of its remote user. A Servlet may retrieve the value of this attribute by calling ServletRequest.getAttribute(HttpContext.AUTHORIZATION). This header name is org.osgi.service.useradmin.authorization.

5. Once the request is authenticated and any attributes are set, the handleSecurity method should return true. This return indicates to the Http Service that the request is authorized and processing may continue. If the request is for a Servlet, the Http Service must then call the service method on the Servlet object.

14.8 Security

This section only applies when executing in an OSGi environment which is enforcing Java permissions.

14.8.1 Accessing Resources in Bundles

The Http Service must be granted AdminPermission so that bundles may use a default HttpContext object. This is necessary because the implementation of the default HttpContext object must call Bundle.getResource to access the resources of a bundle and this method requires the caller to have AdminPermission.

Any bundle may access resources in its own bundle by calling Class.getResource. This operation is privileged. The resulting URL object may then be passed to the Http Service as the result of a HttpContext.getResource call. No further permission checks are performed when accessing bundle resource URL objects, so the Http Service does not need to be granted any additional permissions.

14.8.2 Accessing Other Types of Resources

In order to access resources that were not registered using the default HttpContext object, the Http Service must be granted sufficient privileges to access these resources. For example, if the getResource method of the registered HttpContext object returns a file URL, the Http Service requires the corresponding FilePermission to read the file. Similarly, if the getResource method of the registered HttpContext object returns an HTTP URL, the Http Service requires the corresponding SocketPermission to connect to the resource.

Therefore, in most cases, the Http Service should be a privileged service that is granted sufficient permission to serve any bundle's resources, no matter where these resources are located. Therefore, the Http Service must capture the AccessControlContext object of the bundle registering resources or a servlet, and then use the captured AccessControlContext object when accessing resources returned by the registered HttpContext object. This situation prevents a bundle from registering resources that it does not have permission to access.

Therefore, the Http Service should follow a scheme like the following example. When a resource or servlet is registered, it should capture the context.

```
AccessControlContext acc =
      AccessController.getContext();
```

When a URL returned by the getResource method of the associated HttpContext object is called, the Http Service must call the getResource method in a doPrivileged construct using the AccessControlContext object of the registering bundle:

```
AccessController.doPrivileged(
   new PrivilegedExceptionAction() {
      public Object run() throws Exception {
      ...
      }
   }, acc);
```

The Http Service must only use the captured AccessControlContext when accessing resource URL objects. Servlet and HttpContext objects must use a doPrivileged construct in their implementations when performing privileged operations.

14.9 Configuration Properties

If the Http Service does not have its port values configured through some other means, the Http Service implementation should use the following properties to determine the port values upon which to listen.

The following OSGi environment properties are used to specify default HTTP ports:

- org.osgi.service.http.port – This property specifies the port used for servlets and resources accessible via HTTP. The default value for this property is 80.
- org.osgi.service.http.port.secure – This property specifies the port used for servlets and resources accessible via HTTPS. The default value for this property is 443.

14.10 Changes

The API of the HTTP service has not been changed and the version is therefore also not changed.

14.10.1 Example

The example in *Mapping HTTP Requests to Servlet and Resource Registrations* on page 292 contained two errors in calling non-existing methods. These were corrected.

14.10.2 Use of single /

Ambiguities in the use of a single '/' were corrected and an example was added to Table 18, "Examples of Name-space Mapping," on page 293.

14.10.3 MIME Type Table

Table 19, "Sample Extension to MIME Media Mapping," on page 295 contained the .html extension twice. The first occurrence was replaced with .htm.

14.11 org.osgi.service.http

The OSGi Http Service 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.http; specification-version=1.1
```

14.11.1 Summary

- HttpContext – This interface defines methods that the Http Service may call to get information about a registration. [p.299]
- HttpService – The Http Service allows other bundles in the OSGi environment to dynamically register resources and servlets into the URI namespace of Http Service. [p.301]
- NamespaceException – A NamespaceException is thrown to indicate an error with the caller's request to register a servlet or resources into the URI namespace of the Http Service. [p.303]

14.11.2 public interface HttpContext

This interface defines methods that the Http Service may call to get information about a registration.

Servlets and resources may be registered with an HttpContext object; if no HttpContext object is specified, a default HttpContext object is used. Servlets that are registered using the same HttpContext object will share the same ServletContext object.

This interface is implemented by users of the HttpService.

14.11.2.1 public static final String AUTHENTICATION_TYPE = "org.osgi.service.http.authentication.type"

HttpServletRequest attribute specifying the scheme used in authentication. The value of the attribute can be retrieved by HttpServletRequest.getAuthType. This attribute name is org.osgi.service.http.authentication.type.

Since 1.1

14.11.2.2 public static final String AUTHORIZATION = "org.osgi.service.useradmin.authorization"

HttpServletRequest attribute specifying the Authorization object obtained from the org.osgi.service.useradmin.UserAdmin service. The value of the attribute can be retrieved by HttpServletRequest.getAttribute(HttpContext.AUTHORIZATION). This attribute name is org.osgi.service.useradmin.authorization.

Since 1.1

14.11.2.3 public static final String REMOTE_USER =

"org.osgi.service.http.authentication.remote.user"

`HttpServletRequest` attribute specifying the name of the authenticated user. The value of the attribute can be retrieved by `HttpServletRequest.getRemoteUser`. This attribute name is `org.osgi.service.http.authentication.remote.user`.

Since 1.1

14.11.2.4 public String getMimeType(String name)

name determine the MIME type for this name.

- Maps a name to a MIME type. Called by the Http Service to determine the MIME type for the name. For servlet registrations, the Http Service will call this method to support the `ServletContext` method `getMimeType`. For resource registrations, the Http Service will call this method to determine the MIME type for the Content-Type header in the response.

Returns MIME type (e.g. text/html) of the name or `null` to indicate that the Http Service should determine the MIME type itself.

14.11.2.5 public URL getResource(String name)

name the name of the requested resource

- Maps a resource name to a URL.

  Called by the Http Service to map a resource name to a URL. For servlet registrations, Http Service will call this method to support the `ServletContext` methods `getResource` and `getResourceAsStream`. For resource registrations, Http Service will call this method to locate the named resource. The context can control from where resources come. For example, the resource can be mapped to a file in the bundle's persistent storage area via `bundleContext.getDataFile(name).toURL()` or to a resource in the context's bundle via `getClass().getResource(name)`

Returns URL that Http Service can use to read the resource or `null` if the resource does not exist.

14.11.2.6 public boolean handleSecurity(HttpServletRequest request, HttpServletResponse response) throws IOException

request the HTTP request

response the HTTP response

- Handles security for the specified request.

  The Http Service calls this method prior to servicing the specified request. This method controls whether the request is processed in the normal manner or an error is returned.

  If the request requires authentication and the Authorization header in the request is missing or not acceptable, then this method should set the WWW-Authenticate header in the response object, set the status in the response object to Unauthorized(401) and return `false`. See also RFC 2617: *HTTP Authentication: Basic and Digest Access Authentication* (available at http://www.ietf.org/rfc/rfc2617.txt).

If the request requires a secure connection and the getScheme method in the request does not return 'https' or some other acceptable secure protocol, then this method should set the status in the response object to Forbidden(403) and return false.

When this method returns false, the Http Service will send the response back to the client, thereby completing the request. When this method returns true, the Http Service will proceed with servicing the request.

If the specified request has been authenticated, this method must set the AUTHENTICATION_TYPE[p.299] request attribute to the type of authentication used, and the REMOTE_USER[p.299] request attribute to the remote user (request attributes are set using the setAttribute method on the request). If this method does not perform any authentication, it must not set these attributes.

If the authenticated user is also authorized to access certain resources, this method must set the AUTHORIZATION[p.299] request attribute to the Authorization object obtained from the org.osgi.service.useradmin.UserAdmin service.

The servlet responsible for servicing the specified request determines the authentication type and remote user by calling the getAuthType and getRemoteUser methods, respectively, on the request.

Returns true if the request should be serviced, false if the request should not be serviced and Http Service will send the response back to the client.

Throws IOException – may be thrown by this method. If this occurs, the Http Service will terminate the request and close the socket.

14.11.3 public interface HttpService

The Http Service allows other bundles in the OSGi environment to dynamically register resources and servlets into the URI namespace of Http Service. A bundle may later unregister its resources or servlets.

See Also HttpContext[p.299]

14.11.3.1 public HttpContext createDefaultHttpContext( )

□ Creates a default HttpContext for registering servlets or resources with the HttpService, a new HttpContext object is created each time this method is called.

The behavior of the methods on the default HttpContext is defined as follows:

- getMimeType - Does not define any customized MIME types for the Content-Type header in the response, and always returns null.
- handleSecurity - Performs implementation-defined authentication on the request.
- getResource - Assumes the named resource is in the context bundle; this method calls the context bundle's Bundle.getResource method, and returns the appropriate URL to access the resource. On a Java runtime environment that supports permissions, the Http Service needs to be granted the org.osgi.framework.AdminPermission.

Returns a default HttpContext object.

Since 1.1

14.11.3.2 public void registerResources(String alias, String name, HttpContext context) throws NamespaceException

alias name in the URI namespace at which the resources are registered

name the base name of the resources that will be registered

context the `HttpContext` object for the registered resources, or `null` if a default `HttpContext` is to be created and used.

- Registers resources into the URI namespace.

  The alias is the name in the URI namespace of the Http Service at which the registration will be mapped. An alias must begin with slash ('/') and must not end with slash ('/'), with the exception that an alias of the form "/" is used to denote the root alias. The name parameter must also not end with slash ('/'). See the specification text for details on how HTTP requests are mapped to servlet and resource registrations.

  For example, suppose the resource name /tmp is registered to the alias /files. A request for /files/foo.txt will map to the resource name /tmp/foo.txt.

  ```
  httpservice.registerResources("/files",
    "/tmp",
    context);
  ```

  The Http Service will call the `HttpContext` argument to map resource names to URLs and MIME types and to handle security for requests. If the `HttpContext` argument is `null`, a default `HttpContext` is used (see createDefaultHttpContext[p.301]).

Throws `NamespaceException` – if the registration fails because the alias is already in use.

`IllegalArgumentException` – if any of the parameters are invalid

14.11.3.3 public void registerServlet(String alias, Servlet servlet, Dictionary initparams, HttpContext context) throws ServletException, NamespaceException

alias name in the URI namespace at which the servlet is registered

servlet the servlet object to register

initparams initialization arguments for the servlet or `null` if there are none. This argument is used by the servlet's `ServletConfig` object.

context the `HttpContext` object for the registered servlet, or `null` if a default `HttpContext` is to be created and used.

- Registers a servlet into the URI namespace.

  The alias is the name in the URI namespace of the Http Service at which the registration will be mapped.

  An alias must begin with slash ('/') and must not end with slash ('/'), with the exception that an alias of the form "/" is used to denote the root alias. See the specification text for details on how HTTP requests are mapped to servlet and resource registrations.

  The Http Service will call the servlet's `init` method before returning.

```
httpService.registerServlet("/myservlet",
  servlet,
  initparams,
  context);
```

Servlets registered with the same HttpContext object will share the same ServletContext. The Http Service will call the context argument to support the ServletContext methods getResource, getResourceAsStream and getMimeType, and to handle security for requests. If the context argument is null, a default HttpContext object is used (see createDefaultHttpContext[p.301]).

Throws NamespaceException – if the registration fails because the alias is already in use.

javax.servlet.ServletException – if the servlet's init method throws an exception, or the given servlet object has already been registered at a different alias.

IllegalArgumentException – if any of the arguments are invalid

14.11.3.4 public void unregister(String alias)

alias name in the URI name-space of the registration to unregister

□ Unregisters a previous registration done by registerServlet or registerResources methods.

After this call, the registered alias in the URI name-space will no longer be available. If the registration was for a servlet, the Http Service must call the destroy method of the servlet before returning.

If the bundle which performed the registration is stopped or otherwise "unget"s the Http Service without calling unregister[p.303] then Http Service must automatically unregister the registration. However, if the registration was for a servlet, the destroy method of the servlet will not be called in this case since the bundle may be stopped. unregister[p.303] must be explicitly called to cause the destroy method of the servlet to be called. This can be done in the org.osgi.framework.BundleActivator.stop method of the bundle registering the servlet.

Throws IllegalArgumentException – if there is no registration for the alias or the calling bundle was not the bundle which registered the alias.

14.11.4 public class NamespaceException extends Exception

A NamespaceException is thrown to indicate an error with the caller's request to register a servlet or resources into the URI namespace of the Http Service. This exception indicates that the requested alias already is in use.

14.11.4.1 public NamespaceException(String message)

message the detail message

□ Construct a NamespaceException object with a detail message.

14.11.4.2 public NamespaceException(String message, Throwable exception)

message the detail message

exception the nested exception

- Construct a `NamespaceException` object with a detail message and a nested exception.

14.11.4.3 public Throwable getException( )

- Returns the nested exception.

Returns the nested exception or null if there is no nested exception.

14.12 References

[38] *HTTP 1.0 Specification RFC-1945*
htpp://www.ietf.org/rfc/rfc1945.txt, May 1996

[39] *HTTP 1.1 Specification RFC-2616*
http://www.ietf.org/rfc/rfc2616.txt, June 1999

[40] *Java Servlet Technology*
http://java.sun.com/products/servlet/index.html

[41] *MIME Multipurpose Internet Mail Extension*
http://www.nacs.uci.edu/indiv/ehood/MIME/MIME.html

[42] *Assigned MIME Media Types*
http://www.iana.org/assignments/media-types

[43] *Registration Procedures for new MIME media types*
http://www.ietf.org/rfc/rfc2048.txt

[44] *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*
http://www.ietf.org/rfc/rfc2617.txt

15 Preferences Service Specification

Version 1.0

15.1 Introduction

Many bundles need to save some data persistently--in other words, the data is required to survive the stopping and restarting of the bundle, Framework and OSGi Service Platform. In some cases, the data is specific to a particular user. For example, imagine a bundle that implements some kind of game. User specific persistent data could include things like the user's preferred difficulty level for playing the game. Some data is not specific to a user, which we call *system* data. An example would be a table of high scores for the game.

Bundles which need to persist data in an OSGi environment can use the file system via org.osgi.framework.BundleContext.getDataFile. A file system, however, can store only bytes and characters, and provides no direct support for named values and different data types.

A popular class used to address this problem for Java applications is the java.util.Properties class. This class allows data to be stored as key/value pairs, called *properties*. For example, a property could have a name com.acme.fudd and a value of elmer. The Properties class has rudimentary support for storage and retrieving with its load and store methods. The Properties class, however, has the following limitations:

- Does not support a naming hierarchy.
- Only supports String property values.
- Does not allow its content to be easily stored in a back-end system.
- Has no user name-space management.

Since the Properties class was introduced in Java 1.0, efforts have been undertaken to replace it with a more sophisticated mechanism. One of these efforts is this Preferences Service specification.

15.1.1 Essentials

The focus of this specification is simplicity, not reliable access to stored data. This specification does *not* define a general database service with transactions and atomicity guarantees. Instead, it is optimized to deliver the stored information when needed, but it will return defaults, instead of throwing an exception, when the back-end store is not available. This approach may reduce the reliability of the data, but it makes the service easier to use, and allows for a variety of compact and efficient implementations.

This API is made easier to use by the fact that many bundles can be written to ignore any problems that the Preferences Service may have in accessing the back-end store, if there is one. These bundles will mostly or exclusively use the methods of the Preferences interface which are not declared to throw a BackingStoreException.

This service only supports the storage of scalar values and byte arrays. It is not intended for storing large data objects like documents or images. No standard limits are placed on the size of data objects which can be stored, but implementations are expected to be optimized for the handling of small objects.

A hierarchical naming model is supported, in contrast to the flat model of the Properties class. A hierarchical model maps naturally to many computing problems. For example, maintaining information about the positions of adjustable seats in a car requires information for each seat. In a hierarchy, this information can be modeled as a node per seat.

A potential benefit of the Preferences Service is that it allows user specific preferences data to be kept in a well defined place, so that a user management system could locate it. This benefit could be useful for such operations as cleaning up files when a user is removed from the system, or to allow a user's preferences to be cloned for a new user.

The Preferences Service does *not* provide a mechanism to allow one bundle to access the preferences data of another. If a bundle wishes to allow another bundle to access its preferences data, it can pass a Preferences or PreferencesService object to that bundle.

The Preferences Service is not intended to provide configuration management functionality. For information regarding Configuration Management, refer to the *Configuration Admin Service Specification* on page 181.

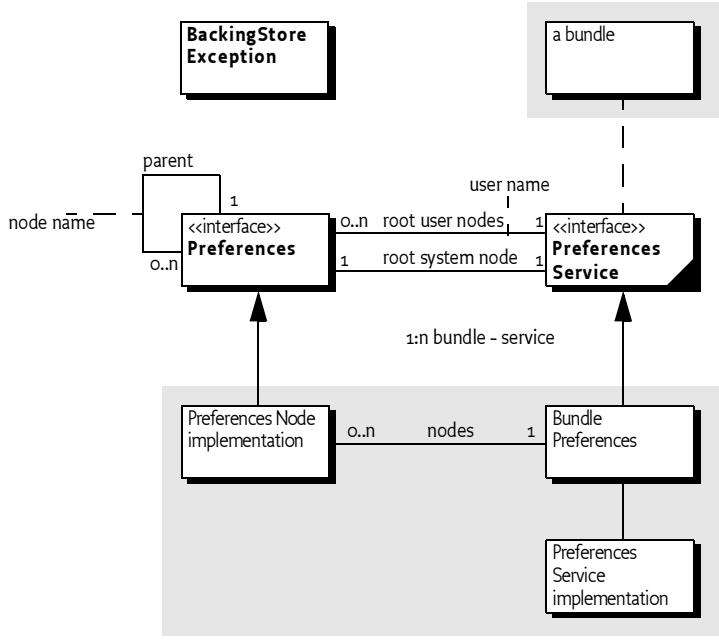
15.1.2 Entities

The PreferencesService is a relatively simple service. It provides access to the different roots of Preferences trees. A single system root node and any number of user root nodes are supported. Each *node* of such a tree is an object that implements the Preferences interface.

This Preferences interface provides methods for traversing the tree, as well as methods for accessing the properties of the node. This interface also contains the methods to flush data into persistent storage, and to synchronize the in-memory data cache with the persistent storage.

All nodes except root nodes have a parent. Nodes can have multiple children.

Figure 48 *Preferences Class Diagram*



15.1.3 Operation

The purpose of the Preferences Service specification is to allow bundles to store and retrieve properties stored in a tree of nodes, where each node implements the Preferences interface. The PreferencesService interface allows a bundle to create or obtain a Preferences tree for system properties, as well as a Preferences tree for each user of the bundle.

This specification allows for implementations where the data is stored locally on the service platform or remotely on a back-end system.

15.2 Preferences Interface

Preferences is an interface that defines the methods to manipulate a node and the tree to which it belongs. A Preferences object contains:

- A set of properties in the form of key/value pairs.
- A parent node.
- A number of child nodes.

15.2.1 Hierarchies

A valid Preferences object always belongs to a *tree*. A tree is identified by its root node. In such a tree, a Preferences object always has a single parent, except for a root node which has a null parent.

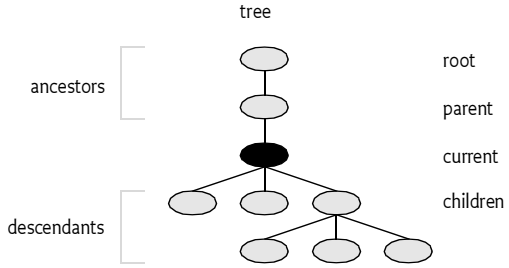
The root node of a tree can be found by recursively calling the parent() method of a node until null is returned. The nodes that are traversed this way are called the *ancestors* of a node.

Each Preferences object has a private name-space for child nodes. Each child node has a name that must be unique among its siblings. Child nodes are created by getting a child node with the node(String) method. The String argument of this call contains a path name. Path names are explained in the next section.

Child nodes can have child nodes recursively. These objects are called the *descendants* of a node.

Descendants are automatically created when they are obtained from a Preferences object, including any intermediate nodes that are necessary for the given path. If this automatic creation is not desired, the nodeExists(String) method can be used to determine if a node already exists.

Figure 49 *Categorization of nodes in a tree*



15.2.2 Naming

Each node has a name relative to its parent. A name may consist of Unicode characters except for the forward slash ("/"). There are no special names, like ".." or ".".

Empty names are reserved for root nodes. Node names that are directly created by a bundle must *always* contain at least one character.

Preferences node names and property keys are *case sensitive*: for example, "org.osgi" and "oRg.oSgI" are two distinct names.

The Preferences Service supports different roots, so there is no absolute root for the Preferences Service. This concept is similar to [46] *Windows Registry* that also supports a number of roots.

A path consists of one or more node names, separated by a slash ("/"). Paths beginning with a "/" are called *absolute path*s while other paths are called *relative paths*. Paths cannot end with a "/" except for the special case of the root node which has absolute path "/".

Path names are always associated with a specific node; this node is called the current node in the following descriptions. Paths identify nodes as follows.

- *Absolute path* – The first "/" is removed from the path, and the remainder of the path is interpreted as a relative path from the tree's root node.
- *Relative path* –
  - If the path is the empty string, it identifies the current node.
  - If the path is a name (does not contain a "/"), then it identifies the child node with that name.

- Otherwise, the first name from the path identifies a child of the current node. The name and slash are then removed from the path, and the remainder of the path is interpreted as a relative path from the child node.

15.2.3 Tree Traversal Methods

A tree can be traversed and modified with the following methods:

- childrenNames() – Returns the names of the child nodes.
- parent() – Returns the parent node.
- removeNode() – Removes this node and all its descendants.
- node(String) – Returns a Preferences object, which is created if it does not already exist. The parameter is an absolute or relative path.
- nodeExists(String) – Returns true if the Preferences object identified by the path parameter exists.

15.2.4 Properties

Each Preferences node has a set of key/value pairs called properties. These properties consist of:

- *Key* – A key is a String object and *case sensitive*.
- The name-space of these keys is separate from that of the child nodes. A Preferences node could have both a child node named fudd and a property named fudd.
- *Value* – A value can always be stored and retrieved as a String object. Therefore, all primitive values must be encoded into String objects. A number of methods are available to store and retrieve values as primitive types. These methods are provided both for the convenience of the user of the Preferences interface, and to allow an implementation the option of storing the values in a more compact form.

All the keys that are defined in a Preferences object can be obtained with the keys() method. The clear() method can be used to clear all properties from a Preferences object. A single property can be removed with the remove(String) method.

15.2.5 Storing and Retrieving Properties

The Preferences interface has a number of methods for storing and retrieving property values based on their key. All the put* methods take as parameters a key and a value. All the get* methods take as parameters a key and a default value.

- put(String,String), get(String,String)
- putBoolean(String,boolean), getBoolean(String,boolean)
- putInt(String,int), getInt(String,int)
- putLong(String,long), getLong(String,long)
- putFloat(String,float), getFloat(String,float)
- putDouble(String,double), getDouble(String,double)
- putByteArray(String,byte[]), getByteArray(String,byte[])

The methods act as if all the values are stored as String objects, even though implementations may use different representations for the different types. For example, a property can be written as a String object and read back as a float, providing that the string can be parsed as a valid Java float object. In the event of a parsing error, the get* methods do not raise exceptions, but instead return their default parameters.

15.2.6 Defaults

All get* methods take a default value as a parameter. The reasons for having such a default are:

- When a property for a Preferences object has not been set, the default is returned instead. In most cases, the bundle developer does not have to distinguish whether or not a property exists.
- A *best effort* strategy has been a specific design choice for this specification. The bundle developer should not have to react when the back-end store is not available. In those cases, the default value is returned without further notice.
  Bundle developers who want to assure that the back-end store is available should call the flush or sync method. Either of these methods will throw a BackingStoreException if the back-end store is not available.

15.3 Concurrency

This specification specifically allows an implementation to modify Preferences objects in a back-end store. If the back-end store is shared by multiple processes, concurrent updates may cause differences between the back-end store and the in-memory Preferences objects.

Bundle developers can partly control this concurrency with the flush() and sync() method. Both methods operate on a Preferences object.

The flush method performs the following actions:

- Stores (makes persistent) any ancestors (including the current node) that do not exist in the persistent store.
- Stores any properties which have been modified in this node since the last time it was flushed.
- Removes from the persistent store any child nodes that were removed from this object since the last time it was flushed.
- Flushes all existing child nodes.

The sync method will first flush, and then ensure that any changes that have been made to the current node and its descendents in the back-end store (by some other process) take effect. For example, it could fetch all the descendants into a local cache, or it could clear all the descendants from the cache so that they will be read from the back-end store as required.

If either method fails, a BackingStoreException is thrown.

The flush or sync methods provide no atomicity guarantee. When updates to the same back-end store are done concurrently by two different processes, the result may be that changes made by different processes are intermingled. To avoid this problem, implementations may simply provide a dedicated section (or name-space) in the back-end store for each OSGi environment, so that clashes do not arise, in which case there is no reason for bundle programmers to ever call sync.

In cases where sync is used, the bundle programmer needs to take into account that changes from different processes may become intermingled, and the level of granularity that can be assumed is the individual property level. Hence, for example, if two properties need to be kept in lockstep, so that one should not be changed without a corresponding change to the other, consider combining them into a single property, which would then need to be parsed into its two constituent parts.

15.4 PreferencesService Interface

The PreferencesService is obtained from the Framework's service registry in the normal way. Its purpose is to provide access to Preferences root nodes.

A Preferences Service maintains a system root and a number of user roots. User roots are automatically created, if necessary, when they are requested. Roots are maintained on a per bundle basis. For example, a user root called elmer in one bundle is distinct from a user root with the same name in another bundle. Also, each bundle has its own system root. Implementations should use a ServiceFactory service object to create a separate PreferencesService object for each bundle.

The precise description of *user* and *system* will vary from one bundle to another. The Preference Service only provides a mechanism, the bundle may use this mechanism in any desired way.

The PreferencesService interface has the following methods to access the system root and user roots:

- getSystemPreferences() – Return a Preferences object that is the root of the system preferences tree.
- getUserPreferences(String) – Return a Preferences object associated with the user name that is given as argument. If the user does not exist, a new root is created atomically.
- getUsers() – Return an array of the names of all the users for whom a Preferences tree exists.

15.5 Cleanup

The Preferences Service must listen for bundle uninstall events, and remove all the preferences data for the bundle that is being uninstalled.

It also must handle the possibility of a bundle getting uninstalled while the Preferences Service is stopped. Therefore, it must check on startup whether preferences data exists for any bundle which is not currently installed. If it does, that data must be removed.

15.6 Changes

- Added several exception clauses to the methods.
- Removed the description of JSR 10 from this specification.

15.7 org.osgi.service.prefs

The OSGi Preferences Service 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.prefs; specification-version=1.0
```

15.7.1 Summary

- BackingStoreException – Thrown to indicate that a preferences operation could not complete because of a failure in the backing store, or a failure to contact the backing store. [p.312]
- Preferences – A node in a hierarchical collection of preference data. [p.312]
- PreferencesService – The Preferences Service. [p.322]

15.7.2 public class BackingStoreException extends Exception

Thrown to indicate that a preferences operation could not complete because of a failure in the backing store, or a failure to contact the backing store.

15.7.2.1 public BackingStoreException(String s)

s the detail message.

□ Constructs a `BackingStoreException` with the specified detail message.

15.7.3 public interface Preferences

A node in a hierarchical collection of preference data.

This interface allows applications to store and retrieve user and system preference data. This data is stored persistently in an implementation-dependent backing store. Typical implementations include flat files, OS-specific registries, directory servers and SQL databases.

For each bundle, there is a separate tree of nodes for each user, and one for system preferences. The precise description of "user" and "system" will vary from one bundle to another. Typical information stored in the user preference tree might include font choice, and color choice for a bundle which interacts with the user via a servlet. Typical information stored in the system preference tree might include installation data, or things like high score information for a game program.

Nodes in a preference tree are named in a similar fashion to directories in a hierarchical file system. Every node in a preference tree has a *node name* (which is not necessarily unique), a unique *absolute path name*, and a path name *relative* to each ancestor including itself.

The root node has a node name of the empty `String` object (""). Every other node has an arbitrary node name, specified at the time it is created. The only restrictions on this name are that it cannot be the empty string, and it cannot contain the slash character ('/').

The root node has an absolute path name of "/". Children of the root node have absolute path names of "/" + *<node name>*. All other nodes have absolute path names of *<parent's absolute path name>* + "/" + *<node name>*. Note that all absolute path names begin with the slash character.

A node *n*'s path name relative to its ancestor *a* is simply the string that must be appended to *a*'s absolute path name in order to form *n*'s absolute path name, with the initial slash character (if present) removed. Note that:

- No relative path names begin with the slash character.
- Every node's path name relative to itself is the empty string.
- Every node's path name relative to its parent is its node name (except for the root node, which does not have a parent).
- Every node's path name relative to the root is its absolute path name with the initial slash character removed.

Note finally that:

- No path name contains multiple consecutive slash characters.
- No path name with the exception of the root's absolute path name end in the slash character.
- Any string that conforms to these two rules is a valid path name.

Each `Preference` node has zero or more properties associated with it, where a property consists of a name and a value. The bundle writer is free to choose any appropriate names for properties. Their values can be of type `String`, `long`, `int`, `boolean`, `byte[]`, `float`, or `double` but they can always be accessed as if they were `String` objects.

All node name and property name comparisons are case-sensitive.

All of the methods that modify preference data are permitted to operate asynchronously; they may return immediately, and changes will eventually propagate to the persistent backing store, with an implementation-dependent delay. The `flush` method may be used to synchronously force updates to the backing store.

Implementations must automatically attempt to flush to the backing store any pending updates for a bundle's preferences when the bundle is stopped or otherwise ungets the Preferences Service.

The methods in this class may be invoked concurrently by multiple threads in a single Java Virtual Machine (JVM) without the need for external synchronization, and the results will be equivalent to some serial execution. If this class is used concurrently *by multiple JVMs* that store their preference data in the same backing store, the data store will not be corrupted, but no other guarantees are made concerning the consistency of the preference data.

15.7.3.1 public String absolutePath( )

- Returns this node's absolute path name. Note that:
  - Root node - The path name of the root node is "/".
  - Slash at end - Path names other than that of the root node may not end in slash ('/').
  - Unusual names - "." and ".." have *no* special significance in path names.
  - Illegal names - The only illegal path names are those that contain multiple consecutive slashes, or that end in slash and are not the root.

Returns this node's absolute path name.

15.7.3.2 public String[] childrenNames( ) throws BackingStoreException

- Returns the names of the children of this node. (The returned array will be of size zero if this node has no children and not null!)

Returns the names of the children of this node.

Throws `BackingStoreException` – if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it.

`IllegalStateException` – if this node (or an ancestor) has been removed with the `removeNode()`[p.322] method.

15.7.3.3 public void clear( ) throws BackingStoreException

- Removes all of the properties (key-value associations) in this node. This call has no effect on any descendants of this node.

Throws `BackingStoreException` – if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it.

`IllegalStateException` – if this node (or an ancestor) has been removed with the `removeNode()`[p.322] method.

See Also `remove(String)`[p.322]

15.7.3.4 public void flush( ) throws BackingStoreException

- Forces any changes in the contents of this node and its descendants to the persistent store.

  Once this method returns successfully, it is safe to assume that all changes made in the subtree rooted at this node prior to the method invocation have become permanent.

  Implementations are free to flush changes into the persistent store at any time. They do not need to wait for this method to be called.

  When a flush occurs on a newly created node, it is made persistent, as are any ancestors (and descendants) that have yet to be made persistent. Note however that any properties value changes in ancestors are *not* guaranteed to be made persistent.

Throws `BackingStoreException` – if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it.

`IllegalStateException` – if this node (or an ancestor) has been removed with the `removeNode()`[p.322] method.

See Also sync() [p.322]

15.7.3.5 public String get(String key, String def)

key key whose associated value is to be returned.

def the value to be returned in the event that this node has no value associated with key or the backing store is inaccessible.

- Returns the value associated with the specified key in this node. Returns the specified default if there is no value associated with the key, or the backing store is inaccessible.

Returns the value associated with key, or def if no value is associated with key.

Throws IllegalStateException – if this node (or an ancestor) has been removed with the removeNode() [p.322] method.

NullPointerException – if key is null. (A null default *is* permitted.)

15.7.3.6 public boolean getBoolean(String key, boolean def)

key key whose associated value is to be returned as a boolean.

def the value to be returned in the event that this node has no value associated with key or the associated value cannot be interpreted as a boolean or the backing store is inaccessible.

- Returns the boolean value represented by the String object associated with the specified key in this node. Valid strings are "true", which represents true, and "false", which represents false. Case is ignored, so, for example, "TRUE" and "False" are also valid. This method is intended for use in conjunction with the putBoolean [p.319] method.

  Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if the associated value is something other than "true" or "false", ignoring case.

Returns the boolean value represented by the String object associated with key in this node, or null if the associated value does not exist or cannot be interpreted as a boolean.

Throws NullPointerException – if key is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode() [p.322] method.

See Also get(String, String) [p.315], putBoolean(String, boolean) [p.319]

15.7.3.7 public byte[] getByteArray(String key, byte[] def)

key key whose associated value is to be returned as a byte [] object.

def the value to be returned in the event that this node has no value associated with key or the associated value cannot be interpreted as a byte [] type, or the backing store is inaccessible.

- Returns the byte[] value represented by the String object associated with the specified key in this node. Valid String objects are *Base64* encoded binary data, as defined in RFC 2045 (http://www.ietf.org/rfc/rfc2045.txt), Section 6.8, with one minor change: the string must consist solely of characters from the *Base64 Alphabet*; no newline characters or extraneous characters are permitted. This method is intended for use in conjunction with the putByteArray[p.320] method.

  Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if the associated value is not a valid Base64 encoded byte array (as defined above).

Returns the byte[] value represented by the String object associated with key in this node, or def if the associated value does not exist or cannot be interpreted as a byte[].

Throws NullPointerException – if key is null. (A null value for def *is* permitted.)

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also get(String, String)[p.315], putByteArray(String, byte[])[p.320]

15.7.3.8 public double getDouble(String key, double def)

key key whose associated value is to be returned as a double value.

def the value to be returned in the event that this node has no value associated with key or the associated value cannot be interpreted as a double type or the backing store is inaccessible.

- Returns the double value represented by the String object associated with the specified key in this node. The String object is converted to an int value as by Double.parseDouble(String). Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if Double.parseDouble(String) would throw a NumberFormatException if the associated value were passed. This method is intended for use in conjunction with the putDouble[p.320] method.

Returns the double value represented by the String object associated with key in this node, or def if the associated value does not exist or cannot be interpreted as a double type.

Throws IllegalStateException – if this node (or an ancestor) has been removed with the the removeNode()[p.322] method.

NullPointerException – if key is null.

See Also putDouble(String, double)[p.320], get(String, String)[p.315]

15.7.3.9 public float getFloat(String key, float def)

key key whose associated value is to be returned as a float value.

def the value to be returned in the event that this node has no value associated with key or the associated value cannot be interpreted as a float type or the backing store is inaccessible.

- Returns the float value represented by the String object associated with the specified key in this node. The String object is converted to an int value as by Float.parseFloat(String). Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if Float.parseFloat(String) would throw a NumberFormatException if the associated value were passed. This method is intended for use in conjunction with the putFloat[p.320] method.

Returns the float value represented by the string associated with key in this node, or def if the associated value does not exist or cannot be interpreted as a float type.

Throws IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

NullPointerException – if key is null.

See Also putFloat(String, float)[p.320], get(String, String)[p.315]

15.7.3.10 public int getInt(String key, int def)

key key whose associated value is to be returned as an int.

def the value to be returned in the event that this node has no value associated with key or the associated value cannot be interpreted as an int or the backing store is inaccessible.

- Returns the int value represented by the String object associated with the specified key in this node. The String object is converted to an int as by Integer.parseInt(String). Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if Integer.parseInt(String) would throw a NumberFormatException if the associated value were passed. This method is intended for use in conjunction with the putInt[p.321] method.

Returns the int value represented by the String object associated with key in this node, or def if the associated value does not exist or cannot be interpreted as an int type.

Throws NullPointerException – if key is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also putInt(String, int)[p.321], get(String, String)[p.315]

15.7.3.11 public long getLong(String key, long def)

key key whose associated value is to be returned as a long value.

def the value to be returned in the event that this node has no value associated with key or the associated value cannot be interpreted as a long type or the backing store is inaccessible.

- Returns the long value represented by the String object associated with the specified key in this node. The String object is converted to a long as by Long.parseLong(String). Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if Long.parseLong(String) would throw a NumberFormatException if the associated value were passed. This method is intended for use in conjunction with the putLong[p.321] method.

Returns the long value represented by the String object associated with key in this node, or def if the associated value does not exist or cannot be interpreted as a long type.

Throws NullPointerException – if key is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also putLong(String, long)[p.321], get(String, String)[p.315]

15.7.3.12 public String[] keys( ) throws BackingStoreException

- Returns all of the keys that have an associated value in this node. (The returned array will be of size zero if this node has no preferences and not null!)

Returns an array of the keys that have an associated value in this node.

Throws BackingStoreException – if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

15.7.3.13 public String name( )

- Returns this node's name, relative to its parent.

Returns this node's name, relative to its parent.

15.7.3.14 public Preferences node(String pathName)

pathName the path name of the Preferences object to return.

- Returns a named Preferences object (node), creating it and any of its ancestors if they do not already exist. Accepts a relative or absolute pathname. Absolute pathnames (which begin with '/') are interpreted relative to the root of this node. Relative pathnames (which begin with any character other than '/') are interpreted relative to this node itself. The empty string ("") is a valid relative pathname, referring to this node itself.

  If the returned node did not exist prior to this call, this node and any ancestors that were created by this call are not guaranteed to become persistent until the flush method is called on the returned node (or one of its descendants).

Returns the specified Preferences object.

Throws IllegalArgumentException – if the path name is invalid.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

NullPointerException – if path name is null.

See Also flush()[p.314]

15.7.3.15 public boolean nodeExists(String pathName) throws BackingStoreException

pathName the path name of the node whose existence is to be checked.

- Returns true if the named node exists. Accepts a relative or absolute pathname. Absolute pathnames (which begin with '/') are interpreted relative to the root of this node. Relative pathnames (which begin with any character other than '/') are interpreted relative to this node itself. The pathname "" is valid, and refers to this node itself.

  If this node (or an ancestor) has already been removed with the removeNode()[p.322] method, it *is* legal to invoke this method, but only with the pathname ""; the invocation will return false. Thus, the idiom p.nodeExists("") may be used to test whether p has been removed.

Returns true if the specified node exists.

Throws BackingStoreException – if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method and pathname is not the empty string ("").

IllegalArgumentException – if the path name is invalid (i.e., it contains multiple consecutive slash characters, or ends with a slash character and is more than one character long).

15.7.3.16 public Preferences parent( )

- Returns the parent of this node, or null if this is the root.

Returns the parent of this node.

Throws IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

15.7.3.17 public void put(String key, String value)

key key with which the specified value is to be associated.

value value to be associated with the specified key.

- Associates the specified value with the specified key in this node.

Throws NullPointerException – if key or value is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

15.7.3.18 public void putBoolean(String key, boolean value)

key key with which the string form of value is to be associated.

value value whose string form is to be associated with key.

- Associates a String object representing the specified boolean value with the specified key in this node. The associated string is "true" if the value is true, and "false" if it is false. This method is intended for use in conjunction with the getBoolean[p.315] method.

  Implementor's note: it is *not* necessary that the value be represented by a string in the backing store. If the backing store supports boolean values, it is not unreasonable to use them. This implementation detail is not visible through the Preferences API, which allows the value to be read as a boolean (with getBoolean) or a String (with get) type.

Throws NullPointerException – if key is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also getBoolean(String, boolean)[p.315], get(String, String)[p.315]

15.7.3.19 public void putByteArray(String key, byte[] value)

key key with which the string form of value is to be associated.

value value whose string form is to be associated with key.

- Associates a String object representing the specified byte[] with the specified key in this node. The associated String object the *Base64* encoding of the byte[], as defined in RFC 2045 (http://www.ietf.org/rfc/rfc2045.txt), Section 6.8, with one minor change: the string will consist solely of characters from the *Base64 Alphabet*; it will not contain any newline characters. This method is intended for use in conjunction with the getByteArray[p.315] method.

  Implementor's note: it is *not* necessary that the value be represented by a String type in the backing store. If the backing store supports byte[] values, it is not unreasonable to use them. This implementation detail is not visible through the Preferences API, which allows the value to be read as an a byte[] object (with getByteArray) or a String object (with get).

Throws NullPointerException – if key or value is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also getByteArray(String, byte[])[p.315], get(String, String)[p.315]

15.7.3.20 public void putDouble(String key, double value)

key key with which the string form of value is to be associated.

value value whose string form is to be associated with key.

- Associates a String object representing the specified double value with the specified key in this node. The associated String object is the one that would be returned if the double value were passed to Double.toString(double). This method is intended for use in conjunction with the getDouble[p.316] method

  Implementor's note: it is *not* necessary that the value be represented by a string in the backing store. If the backing store supports double values, it is not unreasonable to use them. This implementation detail is not visible through the Preferences API, which allows the value to be read as a double (with getDouble) or a String (with get) type.

Throws NullPointerException – if key is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also getDouble(String, double)[p.316]

15.7.3.21 public void putFloat(String key, float value)

key key with which the string form of value is to be associated.

value value whose string form is to be associated with key.

- Associates a String object representing the specified float value with the specified key in this node. The associated String object is the one that would be returned if the float value were passed to Float.toString(float). This method is intended for use in conjunction with the getFloat[p.316] method.

  Implementor's note: it is *not* necessary that the value be represented by a string in the backing store. If the backing store supports float values, it is not unreasonable to use them. This implementation detail is not visible through the Preferences API, which allows the value to be read as a float (with getFloat) or a String (with get) type.

Throws NullPointerException – if key is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also getFloat(String, float)[p.316]

15.7.3.22 public void putInt(String key, int value)

key key with which the string form of value is to be associated.

value value whose string form is to be associated with key.

- Associates a String object representing the specified int value with the specified key in this node. The associated string is the one that would be returned if the int value were passed to Integer.toString(int). This method is intended for use in conjunction with getInt[p.317] method.

  Implementor's note: it is *not* necessary that the property value be represented by a String object in the backing store. If the backing store supports integer values, it is not unreasonable to use them. This implementation detail is not visible through the Preferences API, which allows the value to be read as an int (with getInt or a String (with get) type.

Throws NullPointerException – if key is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also getInt(String, int)[p.317]

15.7.3.23 public void putLong(String key, long value)

key key with which the string form of value is to be associated.

value value whose string form is to be associated with key.

- Associates a String object representing the specified long value with the specified key in this node. The associated String object is the one that would be returned if the long value were passed to Long.toString(long). This method is intended for use in conjunction with the getLong[p.317] method.

  Implementor's note: it is *not* necessary that the value be represented by a String type in the backing store. If the backing store supports long values, it is not unreasonable to use them. This implementation detail is not visible through the Preferences API, which allows the value to be read as a long (with getLong or a String (with get) type.

Throws NullPointerException – if key is null.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also getLong(String, long)[p.317]

15.7.3.24 public void remove(String key)

key key whose mapping is to be removed from this node.

- Removes the value associated with the specified key in this node, if any.

Throws IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also get(String, String)[p.315]

15.7.3.25 public void removeNode( ) throws BackingStoreException

- Removes this node and all of its descendants, invalidating any properties contained in the removed nodes. Once a node has been removed, attempting any method other than name(), absolutePath() or nodeExists("") on the corresponding Preferences instance will fail with an IllegalStateException. (The methods defined on Object can still be invoked on a node after it has been removed; they will not throw IllegalStateException.)

  The removal is not guaranteed to be persistent until the flush method is called on the parent of this node. (It is illegal to remove the root node.)

Throws IllegalStateException – if this node (or an ancestor) has already been removed with the removeNode()[p.322] method.

RuntimeException – if this is a root node.

BackingStoreException – if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it.

See Also flush()[p.314]

15.7.3.26 public void sync( ) throws BackingStoreException

- Ensures that future reads from this node and its descendants reflect any changes that were committed to the persistent store (from any VM) prior to the sync invocation. As a side-effect, forces any changes in the contents of this node and its descendants to the persistent store, as if the flush method had been invoked on this node.

Throws BackingStoreException – if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it.

IllegalStateException – if this node (or an ancestor) has been removed with the removeNode()[p.322] method.

See Also flush()[p.314]

15.7.4 public interface PreferencesService

The Preferences Service.

Each bundle using this service has its own set of preference trees: one for system preferences, and one for each user.

A PreferencesService object is specific to the bundle which obtained it from the service registry. If a bundle wishes to allow another bundle to access its preferences, it should pass its PreferencesService object to that bundle.

15.7.4.1 public Preferences getSystemPreferences( )

- Returns the root system node for the calling bundle.

15.7.4.2 public Preferences getUserPreferences(String name)

- Returns the root node for the specified user and the calling bundle.

15.7.4.3 public String[] getUsers( )

- Returns the names of users for which node trees exist.

15.8 References

[45] *JSR 10 Preferences API*
http://www.jcp.org/jsr/detail/10.jsp

[46] *Windows Registry*
http://www.microsoft.com/technet/win98/reg.asp

[47] *RFC 2045 Base 64 encoding*
http://www.ietf.org/rfc/rfc2045.txt

16 Wire Admin Service Specification

Version 1.0

16.1 Introduction

The Wire Admin service is an administrative service that is used to control a wiring topology in the OSGi Service Platform. It is intended to be used by user interfaces or management programs that control the wiring of services in an OSGi Service Platform.

The Wire Admin service plays a crucial role in minimizing the amount of context-specific knowledge required by bundles when used in a large array of configurations. The Wire Admin service fulfills this role by dynamically *wiring* services together. Bundles participate in this wiring process by registering services that produce or consume data. The Wire Admin service *wires* the services that produce data to services which consume data.

The purpose of wiring services together is to allow configurable cooperation of bundles in an OSGi Service Platform. For example, a temperature sensor can be connected to a heating module to provide a controlled system.

The Wire Admin service is a very important OSGi configuration service and is designed to cooperate closely with the Configuration Admin service, as defined in *Configuration Admin Service Specification* on page 181.

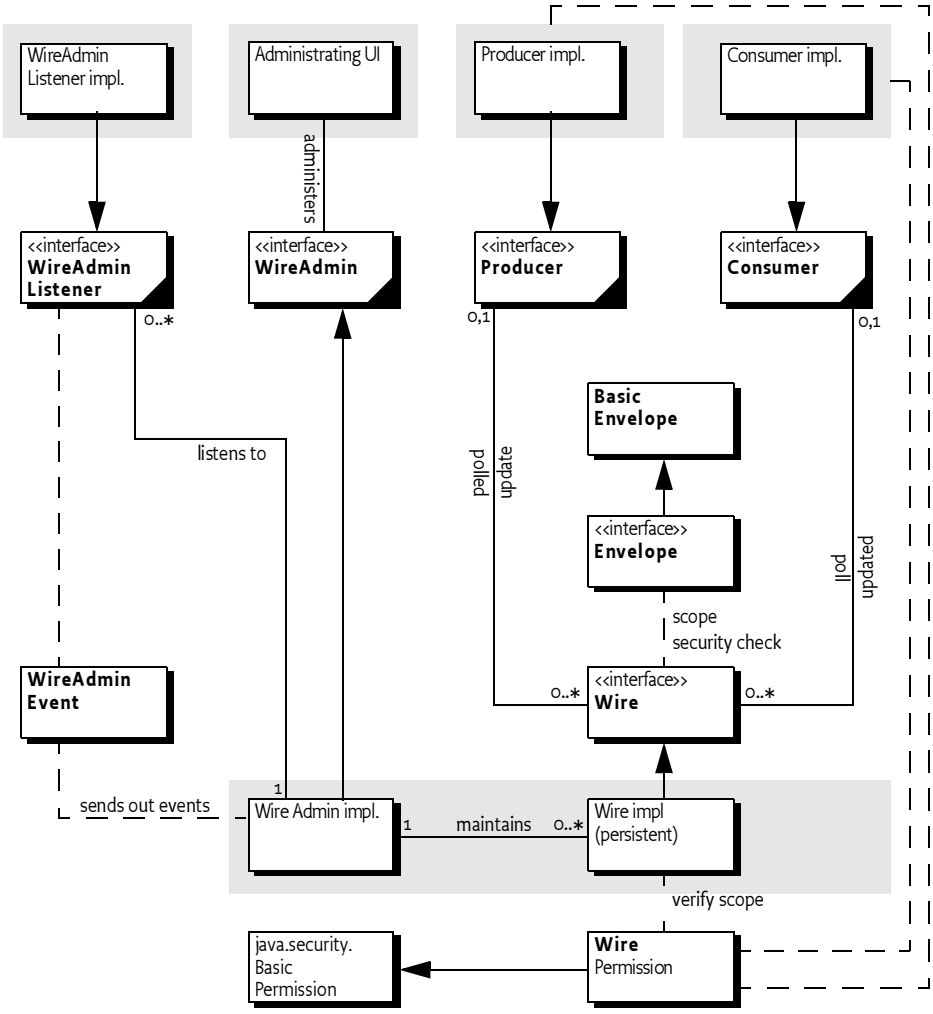
16.1.1 Wire Admin Service Essentials

- *Topology Management* – Provide a comprehensive mechanism to link data-producing components with data-consuming components in an OSGi environment.
- *Configuration Management* – Contains configuration data in order to allow either party to adapt to the special needs of the wire.
- *Data Type Handling* – Facilitate the negotiation of the data type to be used for data transfer between producers of data and consumers of data. Consumers and producers must be able to handle multiple data types for data exchanges using a preferred order.
- *Composites* – Support producers and consumers that can handle a large number of data items.
- *Security* – Separate connected parties from each other. Each party must not be required to hold the service object of the other party.
- *Simplicity* – The interfaces should be designed so that both parties, the Producer and the Consumer services, should be easy to implement.

16.1.2 Wire Admin Service Entities

- *Producer* – A service object that generates information to be used by a Consumer service.
- *Consumer* – A service object that receives information generated by a Producer service.
- *Wire* – An object created by the Wire Admin service that defines an association between a Producer service and a Consumer service. Multiple Wire objects can exist between the same Producer and Consumer pair.
- *WireAdmin* – The service that provides methods to create, update, remove, and list Wire objects.
- *WireAdminListener* – A service that receives events from the Wire Admin service when the Wire object is manipulated or used.
- *WireAdminEvent* – The event that is sent to a WireAdminListener object, describing the details of what happened.
- *Configuration Properties* – Properties that are associated with a Wire object and that contain identity and configuration information set by the administrator of the Wire Admin service.
- *PID* – The Persistent IDentity as defined in the Configuration Admin specification.
- *Flavors* – The different data types that can be used to exchange information between Producer and Consumer services.
- *Composite Producer/Consumer* – A Producer/Consumer service that can generate/accept different kinds of values.
- *Envelope* –An interface for objects that can identify a value that is transferred over the wire. Envelope objects contain also a scope name that is used to verify access permissions.
- *Scope* – A set of names that categorizes the kind of values contained in Envelope objects for security and selection purposes.
- *Basic Envelope* – A concrete implementation of the Envelope interface.
- *WirePermission* – A Permission sub-class that is used to verify if a Consumer service or Producer service has permission for specific scope names.
- *Composite Identity* – A name that is agreed between a composite Consumer and Producer service to identify the kind of objects that they can exchange.

Figure 50 *Class Diagram, org.osgi.service.wiring*



16.1.3 Operation Summary

The Wire Admin service maintains a set of persistent Wire objects. A Wire object contains a Persistent IDentity (PID) for a Consumer service and a PID for a Producer service. (Wire objects can therefore be created when the Producer or Consumer service is not registered.)

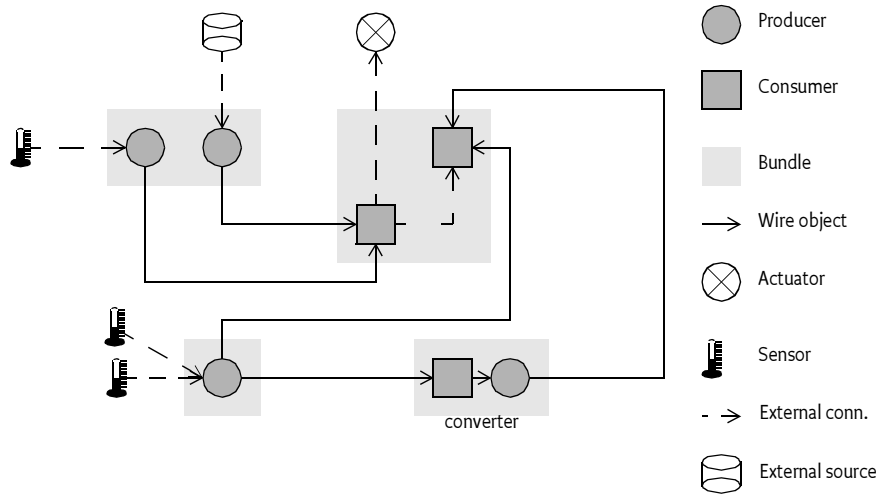
If both those Producer and Consumer services are registered with the Framework, they are connected by the Wire Admin service. The Wire Admin service calls a method on each service object and provides the list of Wire objects to which they are connected.

When a Producer service has new information, it should send this information to each of the connected Wire objects. Each Wire object then must check the filtering and security. If both filtering and security allow the transfer, the Producer service should inform the associated Consumer service with the new information. The Consumer services can also poll a Wire object for an new value at any time.

When a Consumer or Producer service is unregistered from the OSGi Framework, the other object in the association is informed that the Wire object is no longer valid.

Administrative applications can use the Wire Admin service to create and delete wires. These changes are immediately reflected in the current topology and are broadcast to Wire Admin Listener services.

Figure 51 *An Example Wiring Scheme in an OSGi Environment*



16.2 Producer Service

A Producer is a service that can produce a sequence of data objects. For example, a Producer service can produce, among others, the following type of objects:

- Measurement objects that represent a sensor measurement such as temperature, movement, or humidity.
- A String object containing information for user consumption, such as headlines.
- A Date object indicating the occurrence of a periodic event.
- Position information.
- Envelope objects containing status items which can be any type.

16.2.1 Producer Properties

A Producer service must be registered with the OSGi Framework under the interface name org.osgi.service.wireadmin.Producer. The following service properties must be set:

- service.pid – The value of this property, also known as the PID, defines the Persistent IDentity of a service. A Producer service must always use the same PID value whenever it is registered. The PID value allows the Wire Admin service to consistently identify the Producer service and create a persistent Wire object that links a Producer service to a Consumer service. See [48] *Design Patterns* specification for the rules regarding PIDs.
- wireadmin.producer.flavors – The value of this property is an array of Class objects (Class[]) that are the classes of the objects the service can produce. See *Flavors* on page 343 for more information about the data type negotiation between Producer and Consumer services.
- wireadmin.producer.filters – This property indicates to the Wire Admin service that this Producer service performs its own update filtering, meaning that the consumer can limit the number of update calls with a filter expression. This does not modify the data; it only determines whether an update via the wire occurs. If this property is not set, the Wire object must filter according to the description in *Composite objects* on page 335. This service registration property does not need to have a specific value.
- wireadmin.producer.scope – Only for a composite Producer service, a list of scope names that define the scope of this Producer service, as explained in *Scope* on page 336.
- wireadmin.producer.composite – List the composite identities of Consumer services with which this Producer service can interoperate. This property is of type String[]. A composite Consumer service can interoperate with a composite Producer service when there is at least one name that occurs in both the Consumer service's array and the Producer service's array for this property.

16.2.2 Connections

The Wire Admin service connects a Producer service and a Consumer service by creating a Wire object. If the Consumer and Producer services that are bound to a Wire object are registered with the Framework, the Wire Admin service must call the consumersConnected(Wire[]) method on the Producer service object. Every change in the Wire Admin service that affects the Wire object to which a Producer service is connected must result in a call to this method. This requirement ensures that the Producer object is informed of its role in the wiring topology. If the Producer service has no Wire objects attached when it is registered, the Wire Admin service must always call consumersConnected(null). This situation implies that a Producer service can assume it always gets called back from the Wire Admin service when it registers.

16.2.3 Producer Example

The following example shows a clock producer service that sends out a Date object every second.

```
public class Clock extends Thread implements Producer {
  Wire                wires[];
  BundleContext       context;
  boolean             quit;
```

```
    Clock( BundleContext context ) {
       this.context = context;
       start();
    }
    public synchronized void run() {
       Hashtable p = new Hashtable();
       p.put( org.osgi.service.wireadmin.WireConstants.
                 WIREADMIN_PRODUCER_FLAVORS,
              new Class[] { Date.class } );
       p.put( org.osgi.framework.Constants.SERVICE_PID,
          "com.acme.clock" );
       context.registerService(
          Producer.class.getName(),this,p );

       while( ! quit )
       try {
          Date now = new Date();
          for( int i=0; wires!=null && i<wires.length; i++ )
               wires[i].update( now );
          wait( 1000 );
       }
       catch( InterruptedException ie) {
          /* will recheck quit */
       }
    }
    public void synchronized consumersConnected(Wire wires[])
    {
       this.wires = wires;
    }
    public Object polled(Wire wire) { return new Date(); }
   ...
}
```

16.2.4 Push and Pull

Communication between Consumer and Producer services can be initiated in one of the following ways.

- The Producer service calls the update(Object) method on the Wire object. The Wire object implementation must then call the updated(Wire,Object) method on the Consumer service, if the filtering allows this.
- The Consumer service can call poll()on the Wire object. The Wire object must then call polled(Wire) on the Producer object. Update filtering must not apply to polling.

16.2.5 Producers and Flavors

Consumer services can only understand specific data types, and are therefore restricted in what data they can process. The acceptable object classes, the flavors, are communicated by the Consumer service to the Wire Admin service using the Consumer service's service registration properties. The

method getFlavors() on the Wire object returns this list of classes. This list is an ordered list in which the first class is the data type that is the most preferred data type supported by the Consumer service. The last class is the least preferred data type. The Producer service must attempt to convert its data into one of the data types according to the preferred order, or will return null from the poll method to the Consumer service if none of the types are recognized.

Classes cannot be easily compared for equivalence. Sub-classes and interfaces allow classes to masquerade as other classes. The Class.isAssignableFrom(Class) method verifies whether a class is type compatible, as in the following example:

```
Object polled(Wire wire) {
   Class clazzes[] = wire.getFlavors();
   for ( int i=0; i<clazzes.length; i++ ) {
      Class clazz = clazzes[i];
      if ( clazz.isAssignableFrom( Date.class ) )
         return new Date();
      if ( clazz.isAssignableFrom( String.class) )
         return new Date().toString();
   }
   return null;
}
```

The order of the if statements defines the preferences of the Producer object. Preferred data types are checked first. This order normally works as expected but in rare cases, sub-classes can change it. Normally, however, that is not a problem.

16.3 Consumer Service

A Consumer service is a service that receives information from one or more Producer services and is wired to Producer services by the Wire Admin service. Typical Consumer services are as follows:

- The control of an actuator, such as a heating element, oven, or electric shades
- A display
- A log
- A state controller such as an alarm system

16.3.1 Consumer Properties

A Consumer service must be registered with the OSGi Framework under the interface name org.osgi.service.wireadmin.Consumer. The following service properties must be set:

- service.pid – The value of this property, also known as the PID, defines the Persistent IDentity of a service. A Consumer service must always use the same PID value whenever it is registered. The PID value allows the Wire Admin service to consistently identify the Consumer service and create a persistent Wire object that links a Producer service to a Con-

sumer service. See the Configuration Admin specification for the rules regarding PIDs.

- wireadmin.consumer.flavors – The value of this property is an array of Class objects (Class[]) that are the acceptable classes of the objects the service can process. See *Flavors* on page 343 for more information about the data type negotiation between Producer and Consumer services.
- wireadmin.consumer.scope – Only for a composite Consumer service, a list of scope names that define the scope of this Consumer service, as explained in *Scope* on page 336.
- wireadmin.consumer.composite – List the composite identities of Producer services that this Consumer service can interoperate with. This property is of type String[]. A composite Consumer service can interoperate with a composite Producer service when at least one name occurs in both the Consumer service's array and the Producer service's array for this property.

16.3.2 Connections

When a Consumer service is registered and a Wire object exists that associates it to a registered Producer service, the producersConnected(Wire[]) method is called on the Consumer service.

Every change in the Wire Admin service that affects a Wire object to which a Consumer service is connected must result in a call to the producersConnected(Wire[]) method. This rule ensures that the Consumer object is informed of its role in the wiring topology. If the Consumer service has no Wire objects attached, the argument to the producersConnected(Wire[]) method must be null. This method must also be called when a Producer service registers for the first time and no Wire objects are available.

16.3.3 Consumer Example

For example, a service can implement a Consumer service that logs all objects that are sent to it in order to allow debugging of a wiring topology.

```
public class LogConsumer implements Consumer {
   public LogConsumer( BundleContext context ) {
      Hashtable     ht = new Hashtable();
      ht.put(
         Constants.SERVICE_PID, "com.acme.logconsumer" );
      ht.put( WireConstants.WIREADMIN_CONSUMER_FLAVORS,
         new Class[] { Object.class } );
      context.registerService( Consumer.class.getName(),
         this, ht );
   }
   public void updated( Wire wire, Object o ) {
      getLog().log( LogService.LOG_INFO, o.toString() );
   }
   public void producersConnected( Wire [] wires) {}
   LogService getLog() { ... }
}
```

16.3.4 Polling or Receiving a Value

When the Producer service produces a new value, it calls the update(Object) method on the Wire object, which in turn calls the updated(Wire,Object) method on the Consumer service object. When the Consumer service needs a value immediately, it can call the poll() method on the Wire object which in turn calls the polled(Wire) method on the Producer service.

If the poll() method on the Wire object is called and the Producer is unregistered, it must return a null value.

16.3.5 Consumers and Flavors

Producer objects send objects of different data types through Wire objects. A Consumer service object should offer a list of preferred data types (classes) in its service registration properties. The Producer service, however, can still send a null object or an object that is not of the preferred types. Therefore, the Consumer service must check the data type and take the appropriate action. If an object type is incompatible, then a log message should be logged to allow the operator to correct the situation.

The following example illustrates how a Consumer service can handle objects of type Date, Measurement, and String.

```
void process( Object in ) {
  if ( in instanceof Date )
    processDate( (Date) in );
  else if ( in instanceof Measurement )
    processMeasurement( (Measurement) in );
  else if ( in instanceof String )
    processString( (String) in );
  else
    processError( in );
}
```

16.4 Implementation issues

The Wire Admin service can call the consumersConnected or producersConnected methods during the registration of the Consumer or Producer service. Care should be taken in this method call so that no variables are used that are not yet set, such as the ServiceRegistration object that is returned from the registration. The same is true for the updated or polled callback because setting the Wire objects on the Producer service causes such a callback from the consumersConnected or producersConnected method.

A Wire Admin service must call the producersConnected and consumersConnected method asynchronously from the registrations, meaning that the Consumer or Producer service can use synchronized to restrict access to critical variables.

16.5 Wire Properties

A Wire object has a set of properties (a Dictionary object) that configure the association between a Consumer service and a Producer service.

The Wire properties are explained in Table 20.

Constant	Description
WIREADMIN_PID	The value of this property is a unique Persistent IDentity as defined in chapter 10 *Configuration Admin Service Specification*. This PID must be automatically created by the Wire Admin service for each new Wire object.
WIREADMIN_PRODUCER_PID	The value of the property is the PID of the Producer service.
WIREADMIN_CONSUMER_PID	The value of this property is the PID of the Consumer service.
WIREADMIN_FILTER	The value of this property is an OSGi filter string that is used to control the update of produced values. This filter can contain a number of attributes as explained in *Wire Flow Control* on page 339.

Table 20 *Standard Wire Properties*

The properties associated with a Wire object are not limited to the ones defined in Table 20. The Dictionary object can also be used for configuring *both* Consumer services and Producer services. Both services receive the Wire object and can inspect the properties and adapt their behavior accordingly.

16.5.1 Display Service Example

In the following example, the properties of a Wire object, which are set by the Operator or User, are used to configure a Producer service that monitors a user's email account regularly and sends a message when the user has received email. This WireMail service is illustrated as follows:

```
public class WireMail extends Thread
   implements Producer {
   Wire                wires[];
   BundleContext       context;
   boolean             quit;

   public void start( BundleContext context ) {
      Hashtable        ht = new Hashtable();
      ht.put( Constants.SERVICE_PID, "com.acme.wiremail" );
      ht.put( WireConstants.WIREADMIN_PRODUCER_FLAVORS,
          new Class[] { Integer.class } );
      context.registerService( this,
         Producer.class.getName(),
         ht );
   }
```

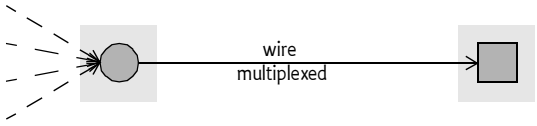
```
    public synchronized void  consumersConnected(
       Wire wires[] ) {
       this.wires = wires;
    }
    public Object polled( Wire wire  ) {
       Dictionary              p = wire.getProperties();
       // The password should be
       // obtained from User Admin Service
       int n = getNrMails(
          p.get( "userid" ),
          p.get( "mailhost" ) );
       return new Integer( n );
    }
    public synchronized void run() {
       while ( !quit )
       try {
          for ( int i=0; wires != null && i<wires.length;i++ )
             wires[i].update( polled( wires[i] ) );

          wait( 150000 );
       }
       catch( InterruptedException e ) { break; }
    }
    ...
}
```

16.6 Composite objects

A Producer and/or Consumer service for each information item is usually the best solution. This solution is not feasible, however, when there are hundreds or thousands of information items. Each registered Consumer or Producer service carries the overhead of the registration, which may overwhelm a Framework implementation on smaller platforms.

When the size of the platform is an issue, a Producer and a Consumer service should abstract a larger number of information items. These Consumer and Producer services are called *composite*.

Figure 52 *Composite Producer Example*



Composite Producer and Consumer services should register respectively the WIREADMIN_PRODUCER_COMPOSITE and WIREADMIN_CONSUMER_COMPOSITE *composite identity* property with their service registration. These properties should contain a list of composite identities. These identities are not defined here, but are up to a mutual agreement between the Consumer and Producer service. For example, a composite identity could be MOST-1.5 or GSM-Phase2-Terminal. The

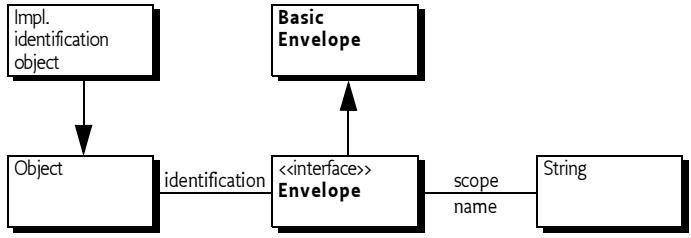
name may follow any scheme but will usually have some version information embedded. The composite identity properties are used to match Consumer and Producer services with each other during configuration of the Wire Admin service. A Consumer and Producer service should interoperate when at least one equal composite identity is listed in both the Producer and Consumer composite identity service property.

Composite producers/consumers must identify the *kind* of objects that are transferred over the Wire object, where *kind* refers to the intent of the object, not the data type. For example, a Producer service can represent the status of a door-lock and the status of a window as a boolean. If the status of the window is transferred as a boolean to the Consumer service, how would it know that this boolean represents the window and not the door-lock?

To avoid this confusion, the Wire Admin service includes an Envelope interface. The purpose of the Envelope interface is to associate a value object with:

- An identification object
- A scope name

Figure 53 *Envelope*



16.6.1 Identification

The Envelope object's identification object is used to identify the value carried in the Envelope object. Each unique kind of value must have its own unique identification object. For example, a left-front-window should have a different identification object than a rear-window.

The identification is of type Object. Using the Object class allows String objects to be used, but also makes it possible to use more complex objects. These objects can convey information in a way that is mutually agreed between the Producer and Consumer service. For example, its type may differ depending on each kind of value so that the *Visitor* pattern, see [48] *Design Patterns*, can be used. Or it may contain specific information that makes the Envelope object easier to dispatch for the Consumer service.

16.6.2 Scope

The scope name is a String object that *categorizes* the Envelope object. The scope name is used to limit the kind of objects that can be exchanged between composite Producer and Consumer services, depending on security settings.

The name-space for this scope should be mutually agreed between the Consumer and Producer services a priori. For the Wire Admin service, the scope name is an opaque string, though its syntax is specified in *Scope name syntax* on page 339.

Both composite Producer and Consumer services must add a list of their supported scope names to the service registration properties. This list is called the *scope* of that service. A Consumer service must add this scope property with the name of WIREADMIN_CONSUMER_SCOPE, a Producer service must add this scope property with the name WIREADMIN_PRODUCER_SCOPE. The type of this property must be a String[] object.

Not registering this property by the Consumer or the Producer service indicates to the Wire Admin service that any Wire object connected to that service must return null for the Wire.getScope() method. This case must be interpreted by the Consumer or Producer service that no scope verification is taking place. Secure Producer services should not produce values for this Wire object and secure Consumer services should not accept values.

It is also allowed to register with a *wildcard*, indicating that all scope names are supported. In that case, the WIREADMIN_SCOPE_ALL (which is String[] { "*" }) should be registered as the scope of the service. The Wire object's scope is then fully defined by the other service connected to the Wire object.

The following example shows how a scope is registered.

```
static String [] scope = { "DoorLock", "DoorOpen", "VIN" };

public void start( BundleContext context ) {
   Dictionary   properties = new Hashtable();
   properties.put(
      WireConstants.WIREADMIN_CONSUMER_SCOPE,
      scope );
   properties.put( WireConstants.WIREADMIN_CONSUMER_PID,
      "com.acme.composite.consumer" );
   context.registerService( Consumer.class.getName(),
      new AcmeConsumer(),
      properties );
}
```

Both a composite Consumer and Producer service must register a scope to receive scope support from the Wire object. These two scopes must be converted into a single Wire object's scope and scope names in this list must be checked for the appropriate permissions. This resulting scope is available from the Wire.getScope() method.

If no scope is set by either the Producer or the Consumer service the result must be null. In that case, the Producer or Consumer service must assume that no security checking is in place. A secure Consumer or Producer service should then refuse to operate with that Wire object.

Otherwise, the resulting scope is the intersection of the Consumer and Producer service scope where each name in the scope, called m, must be implied by a WirePermission[CONSUME,m] of the Consumer service, and WirePermission[PRODUCE,m] of the Producer service.

If either the Producer or Consumer service has registered a wildcard scope then it must not restrict the list of the other service, except for the permission check. If both the Producer and Consumer service registered a wildcard, the resulting list must be WIREADMIN_SCOPE_ALL (String[]{"*"}).

For example, the Consumer service has registered a scope of {A,B,C} and has WirePermission[CONSUME,*]. The Producer service has registered a scope of {B,C,E} and has WirePermission[PRODUCE,C|E]. The resulting scope is then {C}. Table 21 shows this and more examples.

Cs	Cp	Ps	Pp	Wire Scope
null		null		null
{A,B,C}	*	null		null
null		{C,D,E}		null
{A,B,C}	B|C	{A,B,C}	A|B	{B}
*	*	{A,B,C}	A|B|C	{A,B,C}
*	*	*	*	{*}
{A,B,C}	A|B|C	{A,B,C}	X	{}
{A,B,C}	*	{B,C,E}	C|E	{C}

Table 21 *Examples of scope calculation. C=Consumer, P=Producer, p=WirePermission, s=scope*

The Wire object's scope must be calculated only once, when both the Producer and Consumer service become connected. When a Producer or Consumer service subsequently modifies its scope, the Wire object must *not* modify the original scope. A Consumer and a Produce service can thus assume that the scope does not change after the producersConnected method or consumersConnected method has been called.

16.6.3 Access Control

When an Envelope object is used as argument in Wire.update(Object) then the Wire object must verify that the Envelope object's scope name is included in the Wire object's scope. If this is not the case, the update must be ignored (the updated method on the Consumer service must not be called).

A composite Producer represents a number of values, which is different from a normal Producer that can always return a single object from the poll method. A composite Producer must therefore return an array of Envelope objects (Envelope[]). This array must contain Envelope objects for all the values that are in the Wire object's scope. It is permitted to return all possible values for the Producer because the Wire object must remove all Envelope objects that have a scope name not listed in the Wire object's scope.

16.6.4 Composites and Flavors

Composite Producer and Consumer services must always use a flavor of the Envelope class. The data types of the values must be associated with the scope name or identification and mutually agreed between the Consumer and Producer services.

Flavors and Envelope objects both represent categories of different values. Flavors, however, are different Java classes that represent the same kind of value. For example, the tire pressure of the left front wheel could be passed as a Float, an Integer, or a Measurement object. Whatever data type is chosen, it is still the tire pressure of the left front wheel. The Envelope object represents the kind of object, for example the right front wheel tire pressure, or the left rear wheel.

16.6.5 Scope name syntax

Scope names are normal String objects and can, in principle, contain any Unicode character. Scope names are used with the WirePermission class that extends java.security.BasicPermission. The BasicPermission class implements the implies method and performs the name matching. The wildcard matching of this class is based on the concept of names where the constituents of the name are separated with a period ('.'): for example, org.osgi.service.http.port.

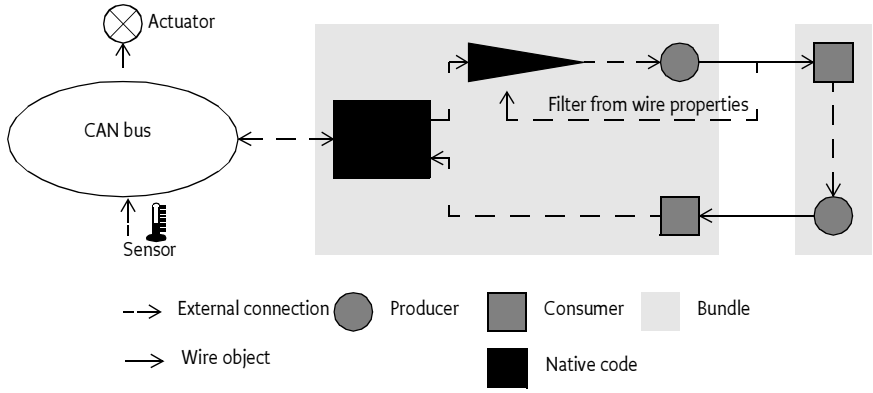
Scope names must therefore follow the rules for fully qualified Java class names. For example, door.lock is a correct scope name while door-lock is not.

16.7 Wire Flow Control

The WIREADMIN_FILTER property contains a filter expression (as defined in the OSGi Framework Filter class, see page 117) that is used to limit the number of updates to the Consumer service. This is necessary because information can arrive at a much greater rate than can be processed by a Consumer service. For example, a single CAN bus (the electronic control bus used in current cars) in a car can easily deliver hundreds of measurements per second to an OSGi based controller. Most of these measurements are not relevant to the OSGi bundles, at least not all the time. For example, a bundle that maintains an indicator for the presence of frost is only interested in measurements when the outside temperature passes the 4 degrees Celsius mark.

Limiting the number of updates from a Producer service can make a significant difference in performance (meaning that less hardware is needed). For example, a vendor can implement the filter in native code and remove unnecessary updates prior to processing in the Java Virtual Machine (JVM). This is depicted in Figure 54 on page 340.

Figure 54 Filtering of Updates



The filter can use any combination of the following attributes in a filter to implement many common filtering schemes:

Constant	Description
WIREVALUE_CURRENT	Current value of the data from the Producer service.
WIREVALUE_PREVIOUS	Previous data value that was reported to the Consumer service.
WIREVALUE_DELTA_ABSOLUTE	The actual positive difference between the previous data value and the current data value. For example, if the previous data value was 3 and the current data value is -0.5, then the absolute delta is 4.5. This filter attribute is not set when the current or previous value is not a number.
WIREVALUE_DELTA_RELATIVE	The absolute (meaning always positive) relative change between the current and the previous data values, calculated with the following formula: |previous-current|/|current|. For example, if the previous value was 3 and the new value is 5, then the relative delta is |3-5|/|5| = 0.4. This filter attribute is not set when the current or previous value is not a number.
WIREVALUE_ELAPSED	The time in milliseconds between the last time the Consumer.updated(Wire,Object) returned and the time the filter is evaluated.

Table 22 Filter Attribute Names

Filter attributes can be used to implement many common filtering schemes that limit the number of updates that are sent to a Consumer service. The Wire Admin service specification requires that updates to a Consumer service are always filtered if the WIREADMIN_FILTER Wire property is present. Producer services that wish to perform the filtering themselves should register with a service property WIREADMIN_PRODUCER_FILTERS. Filtering must be performed by the Wire object for all other Producer services.

Filtering for composite Producer services is not supported. When a filter is set on a Wire object, the Wire must still perform the filtering (which is limited to time filtering because an Envelope object is not a magnitude), but this approach may lose relevant information because the objects are of a different kind. For example, an update of every 500 ms could miss all speed updates because there is a wheel pressure update that resets the elapsed time. Producer services should, however, still implement a filtering scheme that could use proprietary attributes to filter on different kind of objects.

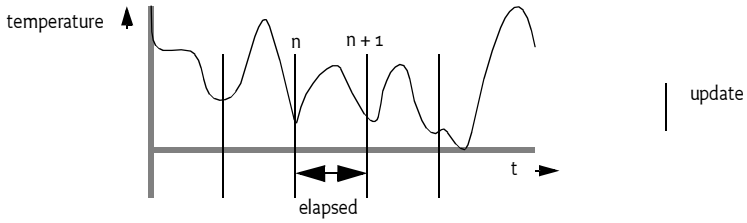
16.7.1 Filtering by Time

The simplest filter mechanism is based on time. The wirevalue.elapsed attribute contains the amount of milliseconds that have passed since the last update to the associated Consumer service. The following example filter expression illustrates how the updates can be limited to approximately 40 times per minute (once every 1500 ms).

```
(wirevalue.elapsed>=1500)
```

Figure 55 depicts this example graphically.

Figure 55 Elapsed Time Change



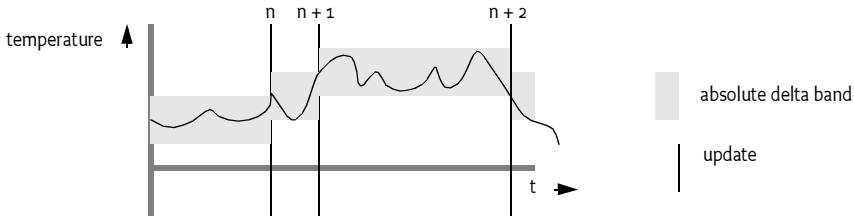
16.7.2 Filtering by Change

A Consumer service is often not interested in an update if the data value has not changed. The following filter expression shows how a Consumer service can limit the updates from a temperature sensor to be sent only when the temperature has changed at least 1 °K.

```
(wirevalue.delta.absolute>=1)
```

Figure 56 depicts a band that is created by the absolute delta between the previous data value and the current data value. The Consumer is only notified with the updated(Wire,Object) method when a data value is outside of this band.

Figure 56 *Absolute Delta*

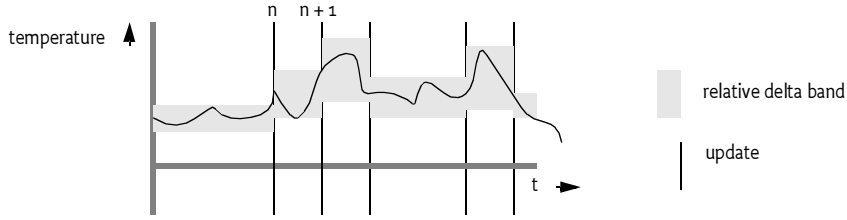


The delta may also be relative. For example, if a car is moving slowly, then updates for the speed of the car are interesting even for small variations. When a car is moving at a high rate of speed, updates are only interesting for larger variations in speed. The following example shows how the updates can be limited to data value changes of at least 10%.

```
(wirevalue.delta.relative>=0.1)
```

Figure 57 on page 342 depicts a relative band. Notice that the size of the band is directly proportional to the size of the sample value.

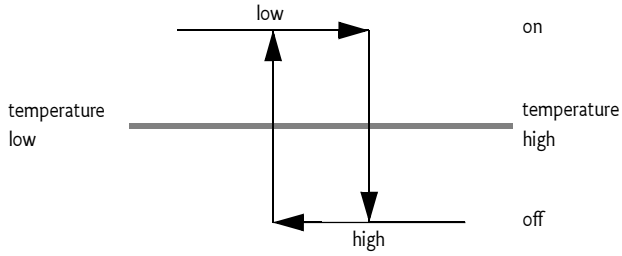
Figure 57 *Relative Delta (not on scale)*



16.7.3 Hysteresis

A thermostat is a control device that usually has a hysteresis, which means that a heater should be switched on below a certain specified low temperature and should be switched off at a specified high temperature, where *high* > *low*. This is graphically depicted in Figure 58 on page 342. The specified acceptable temperatures reduce the amount of start/stops of the heater.

Figure 58 *Hysteresis*



A Consumer service that controls the heater is only interested in events at the top and bottom of the hysteresis. If the specified high value is 250 °K and the specified low value is 249 °K, the following filter illustrates this concept:

```
(| (&(wirevalue.previous<=250) (wirevalue.current>250))
   (&(wirevalue.previous>=249) (wirevalue.current<249))
)
```

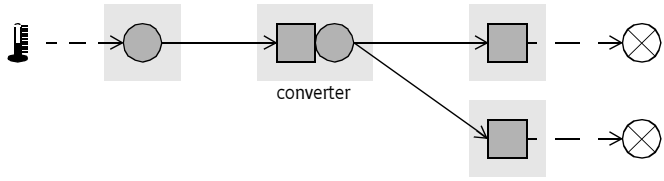
16.8 Flavors

Both Consumer and Producer services should register with a property describing the classes of the data types they can consume or produce respectively. The classes are the *flavors* that the service supports. The purpose of flavors is to allow an administrative user interface bundle to connect Consumer and Producer services. Bundles should only create a connection when there is at least one class shared between the flavors from a Consumer service and a Producer service. Producer services are responsible for selecting the preferred object type from the list of the object types preferred by the Consumer service. If the Producer service cannot convert its data to any of the flavors listed by the Consumer service, null should be used instead.

16.9 Converters

A converter is a bundle that registers a Consumer and a Producer service that are related and performs data conversions. Data values delivered to the Consumer service are processed and transferred via the related Producer service. The Producer service sends the converted data to other Consumer services. This is shown in Figure 59.

Figure 59 *Converter (for legend see Figure 51)*



16.10 Wire Admin Service Implementation

The Wire Admin service is the administrative service that is used to control the wiring topology in the OSGi Service Platform. It contains methods to create or update wires, delete wires, and list existing wires. It is intended to be used by user interfaces or management programs that control the wiring topology of the OSGi Service Platform.

The createWire(String,String,Dictionary) method is used to associate a Producer service with a Consumer service. The method always creates and returns a new object. It is therefore possible to create multiple, distinct wires between a Producer and a Consumer service. The properties can be used to create multiple associations between Producer and Consumer services in that act in different ways.

The properties of a Wire object can be updated with the update(Object) method. This method must update the properties in the Wire object and must notify the associated Consumer and Producer services if they are registered. Wire objects that are no longer needed can be removed with the deleteWire(Wire) method. All these methods are in the WireAdmin class and not in the Wire class for security reasons. See *Security* on page 347.

The getWires(String) method returns an array of Wire objects (or null). All objects are returned when the filter argument is null. Specifying a filter argument limits the returned objects. The filter uses the same syntax as the Framework Filter specification. This filter is applied to the properties of the Wire object and only Wire objects that match this filter are returned.

The following example shows how the getWires method can be used to print the PIDs of Producer services that are wired to a specific Consumer service.

```
String f = "(wireadmin.consumer.pid=com.acme.x)";
Wire [] wires = getWireAdmin().getWires( f );
for ( int i=0; wires != null && i < wires.length; i++ )
   System.out.println(
      wires[i].getProperties().get(
         "wireadmin.producer.pid")
   );
```

16.11 Wire Admin Listener Service Events

The Wire Admin service has an extensive list of events that it can deliver. The events allow other bundles to track changes in the topology as they happen. For example, a graphic user interface program can use the events to show when Wire objects become connected, when these objects are deleted, and when data flows over a Wire object.

A bundle that is interested in such events must register a WireAdminListener service object with a special Integer property WIREADMIN_EVENTS ("wireadmin.events"). This Integer object contains a bitmap of all the events in which this Wire Admin Listener service is interested (events have associated constants that can be OR'd together). A Wire Admin service must not deliver events to the Wire Admin Listener service when that event type is not in the bitmap. If no such property is registered, no events are delivered to the Wire Admin Listener service.

The WireAdminListener interface has only one method: wireAdminEvent(WireAdminEvent). The argument is a WireAdminEvent object that contains the event type and associated data.

A WireAdminEvent object can be sent asynchronously but must be ordered for each Wire Admin Listener service. Wire Admin Listener services must not assume that the state reflected by the event is still true when they receive the event.

The following types are defined for a WireEvent object:

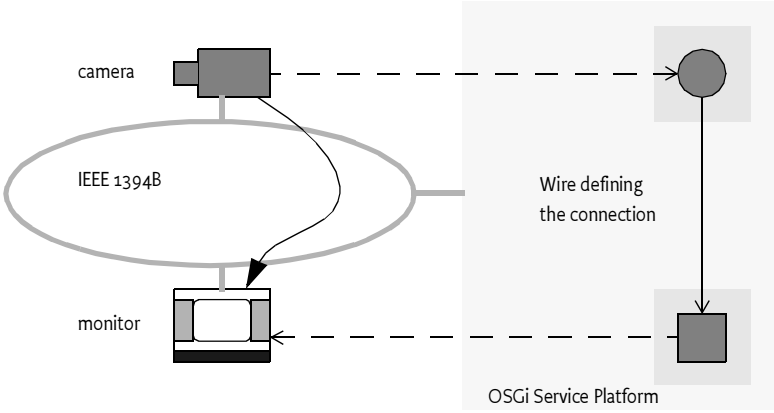
Event type	Description
WIRE_CREATED	A new Wire object has been created.
WIRE_CONNECTED	Both the Producer service and the Consumer service are registered but may not have executed their respective connectedProducers/connectedConsumers methods.
WIRE_UPDATED	The Wire object's properties have been updated.
WIRE_TRACE	The Producer service has called the Wire.update(Object) method with a new value or the Producer service has returned from the Producer.polled(Wire) method.
WIRE_DISCONNECTED	The Producer service or Consumer service have become unregistered and the Wire object is no longer connected.
WIRE_DELETED	The Wire object is deleted from the repository and is no longer available from the getWires method.
CONSUMER_EXCEPTION	The Consumer service generated an exception and the exception is included in the event.
PRODUCER_EXCEPTION	The Producer service generated an exception in a callback and the exception is included in the event.

Table 23 *Events*

16.12 Connecting External Entities

The Wire Admin service can be used to control the topology of consumers and producers that are services, as well as external entities. For example, a video camera controlled over an IEEE 1394B bus can be registered as a Producer service in the Framework's service registry and a TV, also connected to this bus, can be registered as a Consumer service. It would be very inefficient to stream the video data through the OSGi environment. Therefore, the Wire Admin service can be used to supply the external addressing information to the camera and the monitor to make a direct connection *outside* the OSGi environment. The Wire Admin service provides a uniform mechanism to connect both external entities and internal entities.

Figure 60 *Connecting External Entities*



A Consumer service and a Producer service associated with a Wire object receive enough information to establish a direct link because the PIDs of both services are in the Wire object's properties. This situation, however, does not guarantee *compatibility* between Producer and the Consumer service. It is therefore recommended that flavors are used to ensure this compatibility. Producer services that participate in an external addressing scheme, like IEEE 1394B, should have a flavor that reflects this address. In this case, there should then for example be a IEEE 1394B address class. Consumer services that participate in this external addressing scheme should only accept data of this flavor.

The OSGi *Device Access Specification* on page 223, defines the concept of a device category. This is a description of what classes and properties are used in a specific device category: for example, a UPnP device category that defines the interface that must be used to register for a UPnP device, among other things.

Device category descriptions should include a section that addresses the external wiring issue. This section should include what objects are send over the wire to exchange addressing information.

16.13 Related Standards

16.13.1 Java Beans

The Wire Admin service leverages the component architecture that the Framework service registry offers. Java Beans attempt to achieve similar goals. Java Beans are classes that follow a number of recommendations that allow them to be configured at run time. The techniques that are used by Java Beans during configuration are serialization and the construction of adapter classes.

Creating adapter classes in a resource constrained OSGi Service Platform was considered too heavy weight. Also, the dynamic nature of the OSGi environment, where services are registered and unregistered continuously, creates a mismatch between the intended target area of Java Beans and the OSGi Service Platform.

Also, Java Beans can freely communicate once they have a reference to each other. This freedom makes it impossible to control the communication between Java Beans.

This Wire Admin service specification was developed because it is light-weight and leverages the unique characteristics of the OSGi Framework. The concept of a Wire object that acts as an intermediate between the Producer and Consumer service allows the implementation of a security policy because both parties cannot communicate directly.

16.14 Security

16.14.1 Separation of Consumer and Producer Services

The Consumer and Producer service never directly communicate with each other. All communication takes place through a Wire object. This allows a Wire Admin service implementation to control the security aspects of creating a connection, and implies that the Wire Admin service must be a trusted service in a secure environment. Only one bundle should have the ServicePermission[REGISTER,WireAdmin].

ServicePermission[REGISTER,Producer|Consumer] should not be restricted. ServicePermission[GET,Producer|Consumer] must be limited to trusted bundles (the Wire Admin service implementation) because a bundle with this permission can call such services and access information that it should not be able to access.

16.14.2 Using Wire Admin Service

This specification assumes that only a few applications require access to the Wire Admin service. The WireAdmin interface contains all the security sensitive methods that create, update, and remove Wire objects. (This is the reason that the update and delete methods are on the WireAdmin interface and not on the Wire interface). ServicePermission[GET,WireAdmin] should therefore only be given to trusted bundles that can manage the topology.

16.14.3 Wire Permission

Composite Producer and Consumer services can be restricted in their use of scope names. This restriction is managed with the WirePermission class. A WirePermission consists of a scope name and the action CONSUME or PRODUCE. The name used with the WirePermission may contain wild-cards as specified in the java.security.BasicPermission class.

16.15 org.osgi.service.wireadmin

The OSGi Wire Admin service 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.wireadmin; specification-version=1.0
```

16.15.1 Summary

- *BasicEnvelope* – BasicEnvelope is an implementation of the Envelope[p.350] interface [p.348]
- *Consumer* – Data Consumer, a service that can receive udpated values from Producer[p.350] services. [p.348]
- *Envelope* – Identifies a contained value. [p.350]
- *Producer* – Data Producer, a service that can generate values to be used by Consumer[p.348] services. [p.350]
- *Wire* – A connection between a Producer service and a Consumer service. [p.352]
- *WireAdmin* – Wire Administration service. [p.356]
- *WireAdminEvent* – A Wire Admin Event. [p.358]
- *WireAdminListener* – Listener for Wire Admin Events. [p.361]
- *WireConstants* – Defines standard names for Wire properties, wire filter attributes, Consumer and Producer service properties. [p.361]
- *WirePermission* – Permission for the scope of a Wire object. [p.365]

16.15.2 public class BasicEnvelope implements Envelope

BasicEnvelope is an implementation of the Envelope[p.350] interface

16.15.2.1 public BasicEnvelope(Object value, Object identification, String scope)

value Content of this envelope, may be null.

identifying Identifying object for this Envelope object, must not be null

scope Scope name for this object, must not be null

□ Constructor.

See Also Envelope[p.350]

16.15.2.2 public Object getIdentification( )

See Also org.osgi.service.wireadmin.Envelope.getIdentification()[p.350]

16.15.2.3 public String getScope( )

See Also org.osgi.service.wireadmin.Envelope.getScope()[p.350]

16.15.2.4 public Object getValue( )

See Also org.osgi.service.wireadmin.Envelope.getValue()[p.350]

16.15.3 public interface Consumer

Data Consumer, a service that can receive udpated values from Producer[p.350] services.

Service objects registered under the Consumer interface are expected to consume values from a Producer service via a Wire object. A Consumer service may poll the Producer service by calling the Wire.poll[p.355] method. The Consumer service will also receive an updated value when called at it's updated[p.349] method. The Producer service should have coerced the value to be an instance of one of the types specified by the Wire.getFlavors[p.353] method, or one of their subclasses.

Consumer service objects must register with a service.pid and a WireConstants.WIREADMIN_CONSUMER_FLAVORS[p.362] property. It is recommended that Consumer service objects also register with a service.description property.

If an Exception is thrown by any of the Consumer methods, a WireAdminEvent of type WireAdminEvent.CONSUMER_EXCEPTION[p.358] is broadcast by the Wire Admin service.

Security Considerations - Data consuming bundles will require ServicePermission[REGISTER, Consumer]. In general, only the Wire Admin service bundle should have this permission. Thus only the Wire Admin service may directly call a Consumer service. Care must be taken in the sharing of Wire objects with other bundles.

Consumer services must be registered with their scope when they can receive different types of objects from the Producer service. The Consumer service should have WirePermission for each of these scope names.

16.15.3.1 public void producersConnected(Wire[] wires)

wires An array of the current and complete list of Wire objects to which this Consumer service is connected. May be null if the Consumer service is not currently connected to any Wire objects.

- Update the list of Wire objects to which this Consumer service is connected.

  This method is called when the Consumer service is first registered and subsequently whenever a Wire associated with this Consumer service becomes connected, is modified or becomes disconnected.

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

16.15.3.2 public void updated(Wire wire, Object value)

wire The Wire object which is delivering the updated value.

value The updated value. The value should be an instance of one of the types specified by the Wire.getFlavors[p.353] method.

- Update the value. This Consumer service is called by the Wire object with an updated value from the Producer service.

  Note: This method may be called by a Wire object prior to this object being notified that it is connected to that Wire object (via the producersConnected[p.349] method).

When the Consumer service can receive Envelope objects, it must have registered all scope names together with the service object, and each of those names must be permitted by the bundle's WirePermission. If an Envelope object is delivered with the updated method, then the Consumer service should assume that the security check has been performed.

16.15.4 public interface Envelope

Identifies a contained value. An Envelope object combines a status value, an identification object and a scope name. The Envelope object allows the use of standard Java types when a Producer service can produce more than one kind of object. The Envelope object allows the Consumer service to recognize the kind of object that is received. For example, a door lock could be represented by a Boolean object. If the Producer service would send such a Boolean object, then the Consumer service would not know what door the Boolean object represented. The Envelope object contains an identification object so the Consumer service can discriminate between different kinds of values. The identification object may be a simple String object, but it can also be a domain specific object that is mutually agreed by the Producer and the Consumer service. This object can then contain relevant information that makes the identification easier.

The scope name of the envelope is used for security. The Wire object must verify that any Envelope object send through the update method or coming from the poll method has a scope name that matches the permissions of both the Producer service and the Consumer service involved. The wireadmin package also contains a class BasicEnvelope that implements the methods of this interface.

See Also WirePermission[p.365], BasicEnvelope[p.348]

16.15.4.1 public Object getIdentification( )

- Return the identification of this Envelope object. An identification may be of any Java type. The type must be mutually agreed between the Consumer and Producer services.

Returns an object which identifies the status item in the address space of the composite producer, must not be null.

16.15.4.2 public String getScope( )

- Return the scope name of this Envelope object. Scope names are used to restrict the communication between the Producer and Consumer services. Only Envelopes objects with a scope name that is permitted for the Producer and the Consumer services must be passed through a Wire object.

Returns the security scope for the status item, must not be null.

16.15.4.3 public Object getValue( )

- Return the value associated with this Envelope object.

Returns the value of the status item, or null when no item is associated with this object.

16.15.5 public interface Producer

Data Producer, a service that can generate values to be used by Consumer[p.348] services.

Service objects registered under the Producer interface are expected to produce values (internally generated or from external sensors). The value can be of different types. When delivering a value to a Wire object, the Producer service should coerce the value to be an instance of one of the types specified by Wire.getFlavors[p.353]. The classes are specified in order of preference.

When the data represented by the Producer object changes, this object should send the updated value by calling the update method on each of Wire objects passed in the most recent call to this object's consumersConnected[p.351] method. These Wire objects will pass the value on to the associated Consumer service object.

The Producer service may use the information in the Wire object's properties to schedule the delivery of values to the Wire object.

Producer service objects must register with a service.pid and a WireConstants.WIREADMIN_PRODUCER_FLAVORS[p.364] property. It is recommended that a Producer service object also registers with a service.description property. Producer service objects must register with a WireConstants.WIREADMIN_PRODUCER_FILTERS[p.363] property if the Producer service will be performing filtering instead of the Wire object.

If an exception is thrown by a Producer object method, a WireAdminEvent of type WireAdminEvent.PRODUCER_EXCEPTION[p.358] is broadcast by the Wire Admin service.

Security Considerations. Data producing bundles will require ServicePermission[REGISTER, Producer] to register a Producer service. In general, only the Wire Admin service should have ServicePermission[GET, Producer]. Thus only the Wire Admin service may directly call a Producer service. Care must be taken in the sharing of Wire objects with other bundles.

Producer services must be registered with scope names when they can send different types of objects (composite) to the Consumer service. The Producer service should have WirePermission for each of these scope names.

16.15.5.1 public void consumersConnected(Wire[] wires)

wires An array of the current and complete list of Wire objects to which this Producer service is connected. May be null if the Producer is not currently connected to any Wire objects.

□ Update the list of Wire objects to which this Producer object is connected.

This method is called when the Producer service is first registered and subsequently whenever a Wire associated with this Producer becomes connected, is modified or becomes disconnected.

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

16.15.5.2 public Object polled(Wire wire)

wire The `Wire` object which is polling this service.

□ Return the current value of this `Producer` object.

This method is called by a `Wire` object in response to the Consumer service calling the `Wire` object's `poll` method. The Producer should coerce the value to be an instance of one of the types specified by `Wire.getFlavors`[p.353] . The types are specified in order of of preference. The returned value should be as new or newer than the last value furnished by this object.

Note: This method may be called by a `Wire` object prior to this object being notified that it is connected to that `Wire` object (via the `consumersConnected`[p.351] method).

If the Producer service returns an `Envelope` object that has an unpermitted scope name, then the Wire object must ignore (or remove) the transfer.

If the `Wire` object has a scope set, the return value must be an array of `Envelope` objects (`Envelope[]`). The `Wire` object must have removed any `Envelope` objects that have a scope name that is not in the Wire object's scope.

Returns The current value of the Producer service or `null` if the value cannot be coerced into a compatible type. Or an array of `Envelope` objects.

16.15.6 public interface Wire

A connection between a Producer service and a Consumer service.

A `Wire` object connects a Producer service to a Consumer service. Both the Producer and Consumer services are identified by their unique `service.pid` values. The Producer and Consumer services may communicate with each other via `Wire` objects that connect them. The Producer service may send updated values to the Consumer service by calling the `update`[p.355] method. The Consumer service may request an updated value from the Producer service by calling the `poll`[p.355] method.

A Producer service and a Consumer service may be connected through multiple `Wire` objects.

Security Considerations. `Wire` objects are available to Producer and Consumer services connected to a given `Wire` object and to bundles which can access the `WireAdmin` service. A bundle must have `ServicePermission[GET,WireAdmin]` to get the `WireAdmin` service to access all `Wire` objects. A bundle registering a Producer service or a Consumer service must have the appropriate `ServicePermission[REGISTER, Consumer|Producer]` to register the service and will be passed `Wire` objects when the service object's `consumersConnected` or `producersConnected` method is called.

Scope. Each Wire object can have a scope set with the setScope method. This method should be called by a Consumer service when it assumes a Producer service that is composite (supports multiple information items). The names in the scope must be verified by the Wire object before it is used in communication. The semantics of the names depend on the Producer service and must not be interpreted by the Wire Admin service.

16.15.6.1 public Class[] getFlavors( )

- Return the list of data types understood by the Consumer service connected to this Wire object. Note that subclasses of the classes in this list are acceptable data types as well.

  The list is the value of the WireConstants.WIREADMIN_CONSUMER_FLAVORS[p.362] service property of the Consumer service object connected to this object. If no such property was registered or the type of the property value is not Class[], this method must return null.

Returns An array containing the list of classes understood by the Consumer service or null if the Wire is not connected, or the consumer did not register a WireConstants.WIREADMIN_CONSUMER_FLAVORS[p.362] property or the value of the property is not of type Class[].

16.15.6.2 public Object getLastValue( )

- Return the last value sent through this Wire object.

  The returned value is the most recent, valid value passed to the update[p.355] method or returned by the poll[p.355] method of this object. If filtering is performed by this Wire object, this methods returns the last value provided by the Producer service. This value may be an Envelope[] when the Producer service uses scoping. If the return value is an Envelope object (or array), it must be verified that the Consumer service has the proper WirePermission to see it.

Returns The last value passed though this Wire object or null if no valid values have been passed or the Consumer service has no permission.

16.15.6.3 public Dictionary getProperties( )

- Return the wire properties for this Wire object.

Returns The properties for this Wire object. The returned Dictionary must be read only.

16.15.6.4 public String[] getScope( )

- Return the calculated scope of this Wire object. The purpose of the Wire object's scope is to allow a Producer and/or Consumer service to produce/consume different types over a single Wire object (this was deemed necessary for efficiency reasons). Both the Consumer service and the Producer service must set an array of scope names (their scope) with the service registration property WIREADMIN_PRODUCER_SCOPE, or WIREADMIN_CONSUMER_SCOPE when they can produce multiple types. If a Producer service can produce different types, it should set this property to

the array of scope names it can produce, the Consumer service must set the array of scope names it can consume. The scope of a Wire object is defined as the intersection of permitted scope names of the Producer service and Consumer service.

If neither the Consumer, or the Producer service registers scope names with its service registration, then the Wire object's scope must be null.

The Wire object's scope must not change when a Producer or Consumer services modifies its scope.

A scope name is permitted for a Producer service when the registering bundle has WirePermission[PRODUCE], and for a Consumer service when the registering bundle has WirePermission[CONSUME].

If either Consumer service or Producer service has not set a WIREADMIN_*_SCOPE property, then the returned value must be null.

If the scope is set, the Wire object must enforce the scope names when Envelope objects are used as a parameter to update or returned from the poll method. The Wire object must then remove all Envelope objects with a scope name that is not permitted.

Returns A list of permitted scope names or null if the Produce or Consumer service has set no scope names.

16.15.6.5 public boolean hasScope(String name)

name The scope name

- Return true if the given name is in this Wire object's scope.

Returns true if the name is listed in the permitted scope names

16.15.6.6 public boolean isConnected( )

- Return the connection state of this Wire object.

A Wire is connected after the Wire Admin service receives notification that the Producer service and the Consumer service for this Wire object are both registered. This method will return true prior to notifying the Producer and Consumer services via calls to their respective consumersConnected and producersConnected methods.

A WireAdminEvent of type WireAdminEvent.WIRE_CONNECTED[p.359] must be broadcast by the Wire Admin service when the Wire becomes connected.

A Wire object is disconnected when either the Consumer or Producer service is unregistered or the Wire object is deleted.

A WireAdminEvent of type WireAdminEvent.WIRE_DISCONNECTED[p.359] must be broadcast by the Wire Admin service when the Wire becomes disconnected.

Returns true if both the Producer and Consumer for this Wire object are connected to the Wire object; false otherwise.

16.15.6.7 public boolean isValid( )

- Return the state of this Wire object.

A connected Wire must always be disconnected before becoming invalid.

Returns `false` if this `Wire` object is invalid because it has been deleted via WireAdmin.deleteWire[p.357] ; `true` otherwise.

16.15.6.8 public Object poll( )

- Poll for an updated value.

  This methods is normally called by the Consumer service to request an updated value from the Producer service connected to this `Wire` object. This `Wire` object will call the `Producer.polled`[p.352] method to obtain an updated value. If this `Wire` object is not connected, then the Producer service must not be called.

  If this `Wire` object has a scope, then this method must return an array of `Envelope` objects. The objects returned must match the scope of this object. The `Wire` object must remove all `Envelope` objects with a scope name that is not in the `Wire` object's scope. Thus, the list of objects returned must only contain `Envelope` objects with a permitted scope name. If the array becomes empty, `null` must be returned.

  A `WireAdminEvent` of type WireAdminEvent.WIRE_TRACE[p.359] must be broadcast by the Wire Admin service after the Producer service has been successfully called.

Returns A value whose type should be one of the types returned by getFlavors[p.353] , `Envelope[]`, or `null` if the `Wire` object is not connected, the Producer service threw an exception, or the Producer service returned a value which is not an instance of one of the types returned by getFlavors[p.353] .

16.15.6.9 public void update(Object value)

value The updated value. The value should be an instance of one of the types returned by getFlavors[p.353] .

- Update the value.

  This methods is called by the Producer service to notify the Consumer service connected to this `Wire` object of an updated value.

  If the properties of this `Wire` object contain a WireConstants.WIREADMIN_FILTER[p.362] property, then filtering is performed. If the Producer service connected to this `Wire` object was registered with the service property WireConstants.WIREADMIN_PRODUCER_FILTERS[p.363] , the Producer service will perform the filtering according to the rules specified for the filter. Otherwise, this `Wire` object will perform the filtering of the value.

  If no filtering is done, or the filter indicates the updated value should be delivered to the Consumer service, then this `Wire` object must call the `Consumer.updated`[p.349] method with the updated value. If this `Wire` object is not connected, then the Consumer service must not be called and the value is ignored.

  If the value is an `Envelope` object, and the scope name is not permitted, then the `Wire` object must ignore this call and not transfer the object to the Consumer service.

  A `WireAdminEvent` of type WireAdminEvent.WIRE_TRACE[p.359] must be broadcast by the Wire Admin service after the Consumer service has been successfully called.

See Also WireConstants.WIREADMIN_FILTER[p.362]

16.15.7 public interface WireAdmin

Wire Administration service.

This service can be used to create Wire objects connecting a Producer service and a Consumer service. Wire objects also have wire properties that may be specified when a Wire object is created. The Producer and Consumer services may use the Wire object's properties to manage or control their interaction. The use of Wire object's properties by a Producer or Consumer services is optional.

Security Considerations. A bundle must have ServicePermission[GET, WireAdmin] to get the Wire Admin service to create, modify, find, and delete Wire objects.

16.15.7.1 public Wire createWire(String producerPID, String consumerPID, Dictionary properties)

producerPID The service.pid of the Producer service to be connected to the Wire object.

consumerPID The service.pid of the Consumer service to be connected to the Wire object.

properties The Wire object's properties. This argument may be null if the caller does not wish to define any Wire object's properties.

- Create a new Wire object that connects a Producer service to a Consumer service. The Producer service and Consumer service do not have to be registered when the Wire object is created.

  The Wire configuration data must be persistently stored. All Wire connections are reestablished when the WireAdmin service is registered. A Wire can be permanently removed by using the deleteWire[p.357] method.

  The Wire object's properties must have case insensitive String objects as keys (like the Framework). However, the case of the key must be preserved. The type of the value of the property must be one of the following:

```
type        = basetype
 | vector | arrays

basetype = String | Integer | Long
 | Float | Double | Byte
 | Short | Character
 | Boolean

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

arrays   = primitive '[]' | basetype '[]'

vector   = Vector of basetype
```

  The WireAdmin service must automatically add the following Wire properties:

- WireConstants.WIREADMIN_PID[p.363] set to the value of the Wire object's persistent identity (PID). This value is generated by the Wire Admin service when a Wire object is created.
- WireConstants.WIREADMIN_PRODUCER_PID[p.364] set to the value of Producer service's PID.
- WireConstants.WIREADMIN_CONSUMER_PID[p.362] set to the value of Consumer service's PID.

If the properties argument already contains any of these keys, then the supplied values are replaced with the values assigned by the Wire Admin service.

The Wire Admin service must broadcast a WireAdminEvent of type WireAdminEvent.WIRE_CREATED[p.359] after the new Wire object becomes available from getWires[p.357].

Returns The Wire object for this connection.

Throws IllegalArgumentException – If properties contains case variants of the same key name.

16.15.7.2 public void deleteWire(Wire wire)

wire The Wire object which is to be deleted.

□ Delete a Wire object.

The Wire object representing a connection between a Producer service and a Consumer service must be removed. The persistently stored configuration data for the Wire object must destroyed. The Wire object's method Wire.isValid[p.354] will return false after it is deleted.

The Wire Admin service must broadcast a WireAdminEvent of type WireAdminEvent.WIRE_DELETED[p.359] after the Wire object becomes invalid.

16.15.7.3 public Wire[] getWires(String filter) throws InvalidSyntaxException

filter Filter string to select Wire objects or null to select all Wire objects.

□ Return the Wire objects that match the given filter.

The list of available Wire objects is matched against the specified filter. Wire objects which match the filter must be returned. These Wire objects are not necessarily connected. The Wire Admin service should not return invalid Wire objects, but it is possible that a Wire object is deleted after it was placed in the list.

The filter matches against the Wire object's properties including WireConstants.WIREADMIN_PRODUCER_PID[p.364], WireConstants.WIREADMIN_CONSUMER_PID[p.362] and WireConstants.WIREADMIN_PID[p.363].

Returns An array of Wire objects which match the filter or null if no Wire objects match the filter.

Throws InvalidSyntaxException – If the specified filter has an invalid syntax.

See Also org.osgi.framework.Filter

16.15.7.4 public void updateWire(Wire wire, Dictionary properties)

wire The Wire object which is to be updated.

properties The new Wire object's properties or null if no properties are required.

□ Update the properties of a Wire object. The persistently stored configuration data for the Wire object is updated with the new properties and then the Consumer and Producer services will be called at the respective Consumer.producersConnected[p.349] and Producer.consumersConnected[p.351] methods.

The Wire Admin service must broadcast a WireAdminEvent of type WireAdminEvent.WIRE_UPDATED[p.360] after the updated properties are available from the Wire object.

16.15.8 public class WireAdminEvent

A Wire Admin Event.

WireAdminEvent objects are delivered asynchronously to all registered WireAdminListener service objects which specify an interest in the WireAdminEvent type. However, events must be delivered in chronological order with respect to each listener. For example, a WireAdminEvent of type WIRE_CONNECTED[p.359] must be delivered before a WireAdminEvent of type WIRE_DISCONNECTED[p.359] for a particular Wire object.

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

- WIRE_CREATED[p.359]
- WIRE_CONNECTED[p.359]
- WIRE_UPDATED[p.360]
- WIRE_TRACE[p.359]
- WIRE_DISCONNECTED[p.359]
- WIRE_DELETED[p.359]
- PRODUCER_EXCEPTION[p.358]
- CONSUMER_EXCEPTION[p.358]

Event type values must be unique and disjoint bit values. Event types must be defined as a bit in a 32 bit integer and can thus be bitwise OR'ed together.

Security Considerations. WireAdminEvent objects contain Wire objects. Care must be taken in the sharing of Wire objects with other bundles.

See Also WireAdminListener[p.361]

16.15.8.1 public static final int CONSUMER_EXCEPTION = 2

A Consumer service method has thrown an exception.

This WireAdminEvent type indicates that a Consumer service method has thrown an exception. The WireAdminEvent.getThrowable[p.360] method will return the exception that the Consumer service method raised.

The value of CONSUMER_EXCEPTION is 0x00000002.

16.15.8.2 public static final int PRODUCER_EXCEPTION = 1

A Producer service method has thrown an exception.

This WireAdminEvent type indicates that a Producer service method has thrown an exception. The WireAdminEvent.getThrowable[p.360] method will return the exception that the Producer service method raised.

The value of PRODUCER_EXCEPTION is 0x00000001.

16.15.8.3 public static final int WIRE_CONNECTED = 32

The WireAdminEvent type that indicates that an existing Wire object has become connected. The Consumer object and the Producer object that are associated with the Wire object have both been registered and the Wire object is connected. See Wire.isConnected[p.354] for a description of the connected state. This event may come before the producersConnected and consumersConnected method have returned or called to allow synchronous delivery of the events. Both methods can cause other WireAdminEvents to take place and requiring this event to be send before these methods are returned would mandate asynchronous delivery.

The value of WIRE_CONNECTED is 0x00000020.

16.15.8.4 public static final int WIRE_CREATED = 4

A Wire has been created.

This WireAdminEvent type that indicates that a new Wire object has been created. An event is broadcast when WireAdmin.createWire[p.356] is called. The WireAdminEvent.getWire[p.361] method will return the Wire object that has just been created.

The value of WIRE_CREATED is 0x00000004.

16.15.8.5 public static final int WIRE_DELETED = 16

A Wire has been deleted.

This WireAdminEvent type that indicates that an existing wire has been deleted. An event is broadcast when WireAdmin.deleteWire[p.357] is called with a valid wire. WireAdminEvent.getWire[p.361] will return the Wire object that has just been deleted.

The value of WIRE_DELETED is 0x00000010.

16.15.8.6 public static final int WIRE_DISCONNECTED = 64

The WireAdminEvent type that indicates that an existing Wire object has become disconnected. The Consumer object or/and Producer object is/are unregistered breaking the connection between the two. See Wire.isConnected[p.354] for a description of the connected state.

The value of WIRE_DISCONNECTED is 0x00000040.

16.15.8.7 public static final int WIRE_TRACE = 128

The WireAdminEvent type that indicates that a new value is transferred over the Wire object. This event is sent after the Consumer service has been notified by calling the Consumer.updated[p.349] method or the Consumer service requested a new value with the Wire.poll[p.355] method. This is an advisory event meaning that when this event is received, another update may already have occurred and this the Wire.getLastValue[p.353] method returns a newer value then the value that was communicated for this event.

The value of WIRE_TRACE is 0x00000080.

16.15.8.8 public static final int WIRE_UPDATED = 8

A Wire has been updated.

This WireAdminEvent type that indicates that an existing Wire object has been updated with new properties. An event is broadcast when WireAdmin.updateWire[p.357] is called with a valid wire. The WireAdminEvent.getWire[p.361] method will return the Wire object that has just been updated.

The value of WIRE_UPDATED is 0x00000008.

16.15.8.9 public WireAdminEvent(ServiceReference reference, int type, Wire wire, Throwable exception)

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

type The event type. See getType[p.360] .

wire The Wire object associated with this event.

exception An exception associated with this event. This may be null if no exception is associated with this event.

□ Constructs a WireAdminEvent object from the given ServiceReference object, event type, Wire object and exception.

16.15.8.10 public ServiceReference getServiceReference( )

□ Return the ServiceReference object of the Wire Admin service that created this event.

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

16.15.8.11 public Throwable getThrowable( )

□ Returns the exception associated with the event, if any.

Returns An exception or null if no exception is associated with this event.

16.15.8.12 public int getType( )

□ Return the type of this event.

The type values are:

- WIRE_CREATED[p.359]
- WIRE_CONNECTED[p.359]
- WIRE_UPDATED[p.360]

- WIRE_TRACE[p.359]
- WIRE_DISCONNECTED[p.359]
- WIRE_DELETED[p.359]
- PRODUCER_EXCEPTION[p.358]
- CONSUMER_EXCEPTION[p.358]

Returns The type of this event.

16.15.8.13 public Wire getWire( )

□ Return the Wire object associated with this event.

Returns The Wire object associated with this event or null when no Wire object is associated with the event.

16.15.9 public interface WireAdminListener

Listener for Wire Admin Events.

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

WireAdminListener objects can inspect the received WireAdminEvent object to determine its type, the Wire object with which it is associated, and the Wire Admin service that broadcasts the event.

WireAdminListener objects must be registered with a service property WireConstants.WIREADMIN_EVENTS[p.362] whose value is a bitwise OR of all the event types the listener is interested in receiving.

For example:

```
Integer mask = new Integer( WIRE_TRACE
 | WIRE_CONNECTED
 | WIRE_DISCONNECTED );
Hashtable ht = new Hashtable();
ht.put( WIREADMIN_EVENTS, mask );
context.registerService(
   WireAdminListener.class.getName(), this, ht );
```

If a WireAdminListener object is registered without a service property WireConstants.WIREADMIN_EVENTS[p.362], then the WireAdminListener will receive no events.

Security Considerations. Bundles wishing to monitor WireAdminEvent objects will require ServicePermission[REGISTER,WireAdminListener] to register a WireAdminListener service. Since WireAdminEvent objects contain Wire objects, care must be taken in assigning permission to register a WireAdminListener service.

See Also WireAdminEvent[p.358]

16.15.9.1 public void wireAdminEvent(WireAdminEvent event)

event The WireAdminEvent object.

□ Receives notification of a broadcast WireAdminEvent object. The event object will be of an event type specified in this WireAdminListener service's WireConstants.WIREADMIN_EVENTS[p.362] service property.

16.15.10 public interface WireConstants

Defines standard names for `Wire` properties, wire filter attributes, Consumer and Producer service properties.

16.15.10.1 public static final String WIREADMIN_CONSUMER_COMPOSITE = "wireadmin.consumer.composite"

A service registration property for a Consumer service that is composite. It contains the names of the composite Producer services it can cooperate with. Inter-operability exists when any name in this array matches any name in the array set by the Producer service. The type of this property must be `String[]`.

16.15.10.2 public static final String WIREADMIN_CONSUMER_FLAVORS = "wireadmin.consumer.flavors"

Service Registration property (named `wireadmin.consumer.flavors`) specifying the list of data types understood by this Consumer service.

The Consumer service object must be registered with this service property. The list must be in the order of preference with the first type being the most preferred. The value of the property must be of type `Class[]`.

16.15.10.3 public static final String WIREADMIN_CONSUMER_PID = "wireadmin.consumer.pid"

`Wire` property key (named `wireadmin.consumer.pid`) specifying the `service.pid` of the associated Consumer service.

This wire property is automatically set by the Wire Admin service. The value of the property must be of type `String`.

16.15.10.4 public static final String WIREADMIN_CONSUMER_SCOPE = "wireadmin.consumer.scope"

Service registration property key (named `wireadmin.consumer.scope`) specifying a list of names that may be used to define the scope of this `Wire` object. A `Consumer` service should set this service property when it can produce more than one kind of value. This property is only used during registration, modifying the property must not have any effect of the `Wire` object's scope. Each name in the given list mist have `WirePermission[CONSUME]` or else is ignored. The type of this service registration property must be `String[]`.

See Also `Wire.getScope`[p.353], `WIREADMIN_PRODUCER_SCOPE`[p.364]

16.15.10.5 public static final String WIREADMIN_EVENTS = "wireadmin.events"

Service Registration property (named `wireadmin.events`) specifying the `WireAdminEvent` type of interest to a Wire Admin Listener service. The value of the property is a bitwise OR of all the `WireAdminEvent` types the Wire Admin Listener service wishes to receive and must be of type `Integer`.

See Also `WireAdminEvent`[p.358]

16.15.10.6 public static final String WIREADMIN_FILTER = "wireadmin.filter"

`Wire` property key (named `wireadmin.filter`) specifying a filter used to control the delivery rate of data between the Producer and the Consumer service.

This property should contain a filter as described in the `Filter` class. The filter can be used to specify when an updated value from the Producer service should be delivered to the Consumer service. In many cases the Consumer service does not need to receive the data with the same rate that the Producer service can generate data. This property can be used to control the delivery rate.

The filter can use a number of pre-defined attributes that can be used to control the delivery of new data values. If the filter produces a match upon the wire filter attributes, the Consumer service should be notifed of the updated data value.

If the Producer service was registered with the WIREADMIN_PRODUCER_FILTERS[p.363] service property indicating that the Producer service will perform the data filtering then the `Wire` object will not perform data filtering. Otherwise, the `Wire` object must perform basic filtering. Basic filtering includes supporting the following standard wire filter attributes:

- WIREVALUE_CURRENT[p.364] - Current value
- WIREVALUE_PREVIOUS[p.365] - Previous value
- WIREVALUE_DELTA_ABSOLUTE[p.364] - Absolute delta
- WIREVALUE_DELTA_RELATIVE[p.365] - Relative delta
- WIREVALUE_ELAPSED[p.365] - Elapsed time

See Also `org.osgi.framework.Filter`

16.15.10.7 public static final String WIREADMIN_PID = "wireadmin.pid"

`Wire` property key (named `wireadmin.pid`) specifying the persistent identity (PID) of this `Wire` object.

Each `Wire` object has a PID to allow unique and persistent identification of a specific `Wire` object. The PID must be generated by the WireAdmin[p.356] service when the `Wire` object is created.

This wire property is automatically set by the Wire Admin service. The value of the property must be of type `String`.

16.15.10.8 public static final String WIREADMIN_PRODUCER_COMPOSITE = "wireadmin.producer.composite"

A service registration property for a Producer service that is composite. It contains the names of the composite Consumer services it can inter-operate with. Inter-operability exists when any name in this array matches any name in the array set by the Consumer service. The type of this property must be `String[]`.

16.15.10.9 public static final String WIREADMIN_PRODUCER_FILTERS =

"wireadmin.producer.filters"

Service Registration property (named `wireadmin.producer.filters`). A `Producer` service registered with this property indicates to the Wire Admin service that the Producer service implements at least the filtering as described for the WIREADMIN_FILTER[p.362] property. If the Producer service is not registered with this property, the `Wire` object must perform the basic filtering as described in WIREADMIN_FILTER[p.362].

The type of the property value is not relevant. Only its presence is relevant.

16.15.10.10 public static final String WIREADMIN_PRODUCER_FLAVORS = "wireadmin.producer.flavors"

Service Registration property (named `wireadmin.producer.flavors`) specifying the list of data types available from this Producer service.

The Producer service object should be registered with this service property.

The value of the property must be of type `Class[]`.

16.15.10.11 public static final String WIREADMIN_PRODUCER_PID = "wireadmin.producer.pid"

`Wire` property key (named `wireadmin.producer.pid`) specifying the `service.pid` of the associated Producer service.

This wire property is automatically set by the WireAdmin service. The value of the property must be of type `String`.

16.15.10.12 public static final String WIREADMIN_PRODUCER_SCOPE = "wireadmin.producer.scope"

Service registration property key (named `wireadmin.producer.scope`) specifying a list of names that may be used to define the scope of this `Wire` object. A Producer service should set this service property when it can produce more than one kind of value. This property is only used during registration, modifying the property must not have any effect of the `Wire` object's scope. Each name in the given list mist have `WirePermission[PRODUCE, name]` or else is ignored. The type of this service registration property must be `String[]`.

See Also `Wire.getScope`[p.353], WIREADMIN_CONSUMER_SCOPE[p.362]

16.15.10.13 public static final String WIREADMIN_SCOPE_ALL

Matches all scope names.

16.15.10.14 public static final String WIREVALUE_CURRENT = "wirevalue.current"

`Wire` object's filter attribute (named `wirevalue.current`) representing the current value.

16.15.10.15 public static final String WIREVALUE_DELTA_ABSOLUTE = "wirevalue.delta.absolute"

`Wire` object's filter attribute (named `wirevalue.delta.absolute`) representing the absolute delta. The absolute (always positive) difference between the last update and the current value (only when numeric). This attribute must not be used when the values are not numeric.

16.15.10.16 public static final String WIREVALUE_DELTA_RELATIVE = "wirevalue.delta.relative"

`Wire` object's filter attribute (named `wirevalue.delta.relative`) representing the relative delta. The relative difference is (`current` - `previous`) / `current` (only when numeric). This attribute must not be used when the values are not numeric.

16.15.10.17 public static final String WIREVALUE_ELAPSED = "wirevalue.elapsed"

`Wire` object's filter attribute (named `wirevalue.elapsed`) representing the elapsed time, in ms, between this filter evaluation and the last update of the `Consumer` service.

16.15.10.18 public static final String WIREVALUE_PREVIOUS = "wirevalue.previous"

`Wire` object's filter attribute (named `wirevalue.previous`) representing the previous value.

16.15.11 public final class WirePermission extends BasicPermission

Permission for the scope of a `Wire` object. When a `Envelope` object is used for communication with the `poll` or `update` method, and the scope is set, then the `Wire` object must verify that the Consumer service has `WirePermission[name,CONSUME]` and the Producer service has `WirePermission[name,PRODUCE]` for all names in the scope.

The names are compared with the normal rules for permission names. This means that they may end with a "*" to indicate wildcards. E.g. Door.* indicates all scope names starting with the string "Door". The last period is required due to the implementations of the `BasicPermission` class.

16.15.11.1 public static final String CONSUME = "consume"

The action string for the `CONSUME` action: value is "consume".

16.15.11.2 public static final String PRODUCE = "produce"

The action string for the `PRODUCE` action: value is "produce".

16.15.11.3 public WirePermission(String name, String actions)

- Create a new WirePermission with the given name (may be wildcard) and actions.

16.15.11.4 public boolean equals(Object obj)

obj The object to test for equality.

- Determines the equalty of two `WirePermission` objects. Checks that specified object has the same name and actions as this `WirePermission` object.

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

16.15.11.5 public String getActions( )

- Returns the canonical string representation of the actions. Always returns present actions in the following order: `produce`, `consume`.

Returns The canonical string representation of the actions.

16.15.11.6 public int hashCode( )

- Returns the hash code value for this object.

Returns Hash code value for this object.

16.15.11.7 public boolean implies(Permission p)

p The permission to check against.

- Checks if this `WirePermission` object `implies` the specified permission.

  More specifically, this method returns `true` if:

  - *p* is an instanceof the `WirePermission` class,
  - *p*'s actions are a proper subset of this object's actions, and
  - *p*'s name is implied by this object's name. For example, `java.*` implies `java.home`.

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

16.15.11.8 public PermissionCollection newPermissionCollection( )

- Returns a new `PermissionCollection` object for storing `WirePermission` objects.

Returns A new `PermissionCollection` object suitable for storing `WirePermission` objects.

16.15.11.9 public String toString( )

- Returns a string describing this `WirePermission`. The convention is to specify the class name, the permission name, and the actions in the following format: '(org.osgi.service.wireadmin.WirePermission "name" "actions")'.

Returns information about this `Permission` object.

16.16 References

[48] *Design Patterns*
Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Addison Wesley, ISBN 0-201-63361

17 XML Parser Service Specification

Version 1.0

17.1 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 [52] *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

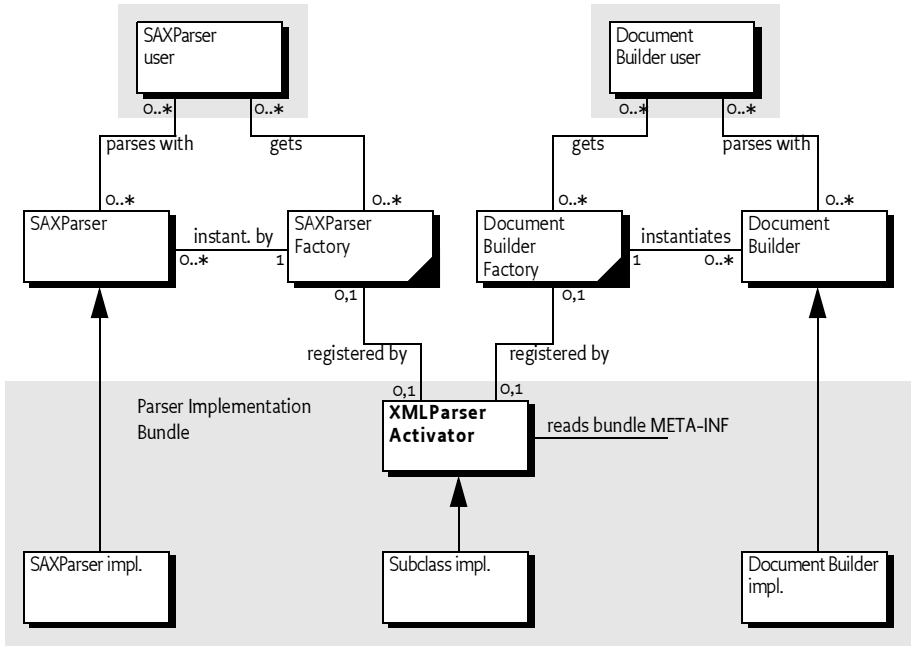
17.1.1 Essentials

- *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

17.1.2 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 61 XML Parsing diagram



17.1.3 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.

17.2 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 [52] *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.

17.3 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.

17.4 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.

17.5 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;
   return (DocumentBuilder) context.getService(ref);
}
```

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;
   return (SAXParser) context.getService(refs[0]);
}
```

17.6 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.

17.6.1 JAR Based Services

Its functionality is based on the definition of the [53] *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.

17.6.2 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 369, that describe the instances.

17.6.3 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.

17.7 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 [52] *JAXP* for the exact version dependencies.

This specification is related to related packages as defined in the JAXP 1.1 document. Table 24 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 24 JAXP 1.1 minimum package versions

The Xerces project from the Apache group, [54] *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.

17.8 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.

17.9 org.osgi.util.xml

The OSGi XML Parser service Package. Specification Version 1.0.

17.9.1 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.374])
- `javax.xml.parsers.DocumentBuilderFactory`(DOMFACTORYNAME[p.373])

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.374] - indicates if this factory supports validating parsers. It's value is a `Boolean`.
- `PARSER_NAMESPACEAWARE`[p.374] - 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.

17.9.1.1 public static final String DOMCLASSFILE = "/META-INF/services/javax.xml.parsers.DocumentBuilderFactory"

Fully qualified path name of DOM Parser Factory Class Name file

17.9.1.2 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.

17.9.1.3 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.

17.9.1.4 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.

17.9.1.5 public static final String SAXCLASSFILE = "/META-INF/services/ javax.xml.parsers.SAXParserFactory"

Fully qualified path name of SAX Parser Factory Class Name file

17.9.1.6 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.

17.9.1.7 public XMLParserActivator( )

17.9.1.8 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

17.9.1.9 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);

17.9.1.10 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);

17.9.1.11 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`

17.9.1.12 public void stop(BundleContext context) throws Exception

context The execution context of the bundle being stopped.

- This method has nothing to do as all open 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`

17.9.1.13 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.

17.10 References

[49] *XML*
http://www.w3.org/XML

[50] *SAX*
http://www.saxproject.org/

[51] *DOM Java Language Binding*
http://www.w3.org/TR/REC-DOM-Level-1/java-language-binding.html

[52] *JAXP*
http://java.sun.com/xml/jaxp

[53] *JAR File specification, services directory*
http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html

[54] *Xerces 2 Java Parser*
http://xml.apache.org/xerces2-j

18 Metatype Specification

Version 1.0

18.1 Introduction

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

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

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

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

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

As an example, the OSGi Service Platform Specifications define several attribute types which are used in a Framework implementation, but which are also used and referenced by other OSGi specifications such as the *Configuration Admin Service Specification* on page 181. A Configuration Admin service implementation deploys attributes (key/value pairs) as configuration properties.

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

18.1.1 Essentials

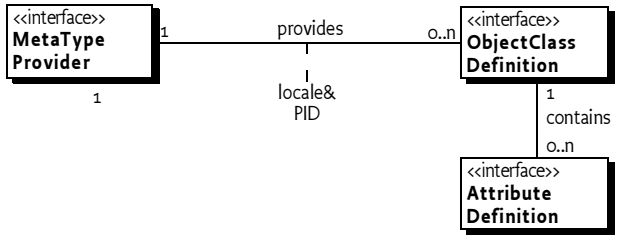
- *Conceptual model* – The specification must have a conceptual model for how classes and attributes are organized.

- *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.

18.1.2 Entities

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

Figure 62 *Class Diagram Meta Typing, org.osgi.service.metatyping*



18.1.3 Operation

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

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

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

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

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

18.2 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 [56] *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.

18.3 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.

18.4 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.

18.5 Meta Type Provider

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

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

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

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

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

18.6 Metatype Example

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

18.7 Limitations

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

18.8 Related Standards

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

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

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

18.8.1 Beans

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

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

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

18.9 Security Considerations

Special security issues are not applicable for this specification.

18.10 Changes

This specification has not been changed since the previous release.

18.11 org.osgi.service.metatype

The OSGi Metatype 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.metatype; specification-version=1.0
```

18.11.1 Summary

- AttributeDefinition – An interface to describe an attribute. [p.384]
- MetaTypeProvider – Provides access to metatypes. [p.387]
- ObjectClassDefinition – Description for the data type information of an objectclass. [p.379]

18.11.2 public interface AttributeDefinition

An interface to describe an attribute.

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

18.11.2.1 public static final int BIGDECIMAL = 10

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

18.11.2.2 public static final int BIGINTEGER = 9

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

18.11.2.3 public static final int BOOLEAN = 11

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

18.11.2.4 public static final int BYTE = 6

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

18.11.2.5 public static final int CHARACTER = 5

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

18.11.2.6 public static final int DOUBLE = 7

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

18.11.2.7 public static final int FLOAT = 8

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

18.11.2.8 public static final int INTEGER = 3

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

18.11.2.9 public static final int LONG = 2

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

18.11.2.10 public static final int SHORT = 4

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

18.11.2.11 public static final int STRING = 1

The STRING(1) type.

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

18.11.2.12 public int getCardinality( )

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

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

18.11.2.13 public String[] getDefaultValue( )

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

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

18.11.2.14 public String getDescription( )

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

Returns The localized description of the definition.

18.11.2.15 public String getID( )

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

Returns The id or oid

18.11.2.16 public String getName( )

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

Returns The localized name of the definition.

18.11.2.17 public String[] getOptionLabels( )

- Return a list of labels of option values.

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

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

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

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

Returns A list values

18.11.2.18 public String[] getOptionValues( )

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

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

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

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

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

Returns A list values

18.11.2.19 public int getType( )

- Return the type for this attribute.

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

18.11.2.20 public String validate(String value)

value The value before turning it into the basic data type

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

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

Returns `null`, "", or another string

18.11.3 public interface MetaTypeProvider

Provides access to metatypes.

18.11.3.1 public String[] getLocales( )

- Return a list of locales available or null if only 1 The return 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.

18.11.3.2 public ObjectClassDefinition getObjectClassDefinition(String pid, String locale)

pid The PID for which the type is needed or null if there is only 1

locale The locale of the definition or null for default locale

- Return the definition of this object class for a 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.

  The implementation should use the locale parameter to match an ObjectClassDefinition object. It should follow the customary locale search path by removing the latter parts of the name.

Returns the ObjectClassDefinition object

18.11.4 public interface ObjectClassDefinition

Description for the data type information of an objectclass.

18.11.4.1 public static final int ALL = -1

Argument for getAttributeDefinitions (int). ALL indicates that all the definitions are returned. The value is -1.

18.11.4.2 public static final int OPTIONAL = 2

Argument for getAttributeDefinitions (int). OPTIONAL indicates that only the optional definitions are returned. The value is 2.

18.11.4.3 public static final int REQUIRED = 1

Argument for getAttributeDefinitions (int). REQUIRED indicates that only the required definitions are returned. The value is 1.

18.11.4.4 public AttributeDefinition[] getAttributeDefinitions(int filter)

filter ALL, REQUIRED, OPTIONAL

- Return the attribute definitions.

  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

18.11.4.5 public String getDescription( )

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

Returns The localized description of the definition.

18.11.4.6 public InputStream getIcon(int size) throws IOException

sizeHint 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`

18.11.4.7 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 or oid

18.11.4.8 public String getName( )

- Return the name of this class.

Returns The name of the described class.

18.12 References

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

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

19 Service Tracker Specification

Version 1.2

19.1 Introduction

The Framework provides a powerful and very dynamic programming environment. Bundles are installed, started, stopped, updated, and uninstalled without shutting down the Framework. Dependencies between bundles are monitored by the Framework, but bundles *must* cooperate in handling these dependencies correctly.

An important aspect of the Framework is the service registry. Bundle developers must be careful not to use service objects that have been unregistered. The dynamic nature of the Framework service registry makes it necessary to track the service objects as they are registered and unregistered. It is easy to overlook rare race conditions or boundary conditions that will lead to random errors.

An example of a potential problem is what happens when the initial list of services of a certain type is created when a bundle is started. When the ServiceListener object is registered before the Framework is asked for the list of services, without special precautions, duplicates can enter the list. When the ServiceListener object is registered after the list is made, it is possible to miss relevant events.

The specification defines a utility class, ServiceTracker, that makes tracking the registration, modification, and unregistration of services much easier. A ServiceTracker class can be customized by implementing the interface or by sub-classing the ServiceTracker class.

This utility specifies a class that significantly reduces the complexity of tracking services in the service registry.

19.1.1 Essentials

- *Customizable* – Allow a default implementation to be customized so that bundle developers can start simply and later extend the implementation to meet their needs.
- *Small* – Every Framework implementation should have this utility implemented. It should therefore be very small because some Framework implementations target minimal OSGi Service Platforms.
- *Tracked set* – Track a single object defined by a ServiceReference object, all instances of a service, or any set specified by a filter expression.

19.1.2 Operation

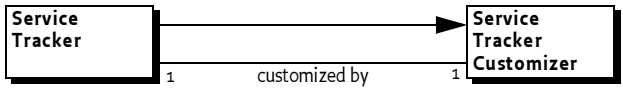
The fundamental tasks of a ServiceTracker object are:

- To create an initial list of services as specified by its creator.
- To listen to ServiceEvent instances so that services of interest to the owner are properly tracked.
- To allow the owner to customize the tracking process through programmatic selection of the services to be tracked, as well as to act when a service is added or removed.

A ServiceTracker object populates a set of services that match a given search criteria, and then listens to ServiceEvent objects which correspond to those services.

19.1.3 Entities

Figure 63 *Class diagram of org.osgi.util.tracker*



19.1.4 Prerequisites

This specification requires OSGi Framework version 1.1 or higher because the Service Tracker uses the Filter class that was not available in version 1.0.

19.2 ServiceTracker Class

The ServiceTracker interface defines three constructors to create ServiceTracker objects, each providing different search criteria:

- ServiceTracker(BundleContext,String,ServiceTrackerCustomizer) – This constructor takes a service interface name as the search criterion. The ServiceTracker object must then track all services that are registered under the specified service interface name.
- ServiceTracker(BundleContext,Filter,ServiceTrackerCustomizer) – This constructor uses a Filter object to specify the services to be tracked. The ServiceTracker must then track all services that match the specified filter.
- ServiceTracker(BundleContext,ServiceReference,ServiceTrackerCustomizer) – This constructor takes a ServiceReference object as the search criterion. The ServiceTracker must then track only the service that corresponds to the specified ServiceReference. Using this constructor, no more than one service must ever be tracked, because a ServiceReference refers to a specific service.

Each of the ServiceTracker constructors takes a BundleContext object as a parameter. This BundleContext object must be used by a ServiceTracker object to track, get, and unget services.

A new ServiceTracker object must not begin tracking services until its open method is called.

19.3 Using a Service Tracker

Once a ServiceTracker object is opened, it begins tracking services immediately. A number of methods are available to the bundle developer to monitor the services that are being tracked. The ServiceTracker class defines these methods:

- getService() – Returns one of the services being tracked or null if there are no active services being tracked.
- getServices() – Returns an array of all the tracked services. The number of tracked services is returned by the size method.
- getServiceReference() – Returns a ServiceReference object for one of the services being tracked. The service object for this service may be returned by calling the ServiceTracker object's getService() method.
- getServiceReferences() – Returns a list of the ServiceReference objects for services being tracked. The service object for a specific tracked service may be returned by calling the ServiceTracker object's getService(ServiceReference) method.
- waitForService(long) – Allows the caller to wait until at least one instance of a service is tracked or until the time-out expires. If the time-out is zero, the caller must wait until at least one instance of a service is tracked. waitForService must not used within the BundleActivator methods, as these methods are expected to complete in a short period of time. A Framework could wait for the start method to complete before starting the bundle that registers the service for which the caller is waiting, creating a deadlock situation.
- remove(ServiceReference) – This method may be used to remove a specific service from being tracked by the ServiceTracker object, causing removedService to be called for that service.
- close() – This method must remove all services being tracked by the ServiceTracker object, causing removedService to be called for all tracked services.
- getTrackingCount() – A Service Tracker can have services added, modified, or removed at any moment in time. The getTrackingCount method is intended to efficiently detect changes in a Service Tracker. Every time the Service Tracker is changed, it must increase the tracking count. A method that processes changes in a Service Tracker could get the tracking count before it processes the changes. If the tracking count has changed at the end of the method, the method should be repeated because a new change occurred during processing.

19.4 Customizing the ServiceTracker class

The behavior of the ServiceTracker class can be customized either by providing a ServiceTrackerCustomizer object implementing the desired behavior when the ServiceTracker object is constructed, or by sub-classing the ServiceTracker class and overriding the ServiceTrackerCustomizer methods.

The ServiceTrackerCustomizer interface defines these methods:

- addingService(ServiceReference) – Called whenever a service is being added to the ServiceTracker object.
- modifiedService(ServiceReference,Object) – Called whenever a tracked service is modified.
- removedService(ServiceReference,Object) – Called whenever a tracked service is removed from the ServiceTracker object.

When a service is being added to the ServiceTracker object or when a tracked service is modified or removed from the ServiceTracker object, it must call addingService, modifiedService, or removedService, respectively, on the ServiceTrackerCustomizer object (if specified when the ServiceTracker object was created); otherwise it must call these methods on itself.

A bundle developer may customize the action when a service is tracked. Another reason for customizing the ServiceTracker class is to programmatically select which services are tracked. A filter may not sufficiently specify the services that the bundle developer is interested in tracking. By implementing addingService, the bundle developer can use additional runtime information to determine if the service should be tracked. If null is returned by the addingService method, the service must not be tracked.

Finally, the bundle developer can return a specialized object from addingService that differs from the service object. This specialized object could contain the service object and any associated information. This returned object is then tracked instead of the service object. When the removedService method is called, the object that is passed along with the ServiceReference object is the one that was returned from the earlier call to the addingService method.

19.4.1 Symmetry

If sub-classing is used to customize the Service Tracker, care must be exercised in using the default implementations of the addingService and removedService methods. The addingService method will get the service and the removedService method assumes it has to unget the service. Overriding one and not the other may thus cause unexpected results.

19.5 Customizing Example

An example of customizing the action taken when a service is tracked might be registering a Servlet object with each Http Service that is tracked. This customization could be done by sub-classing the ServiceTracker class and overriding the addingService and removedService methods as follows:

```
public Object addingService( ServiceReference reference) {
   Object obj = context.getService(reference);
   HttpService svc = (HttpService)obj;
   // Register the Servlet using svc
   ...
   return svc;
}
```

```
public void removedService( ServiceReference reference,
   Object obj ){
   HttpService svc = (HttpService)obj;
   // Unregister the Servlet using svc
   ...
   context.ungetService(reference);
}
```

19.6 Security

A ServiceTracker object contains a BundleContext instance variable that is accessible to the methods in a subclass. A BundleContext object should never be given to other bundles because it is used for security aspects of the Framework.

The ServiceTracker implementation does not have a method to get the BundleContext object but subclasses should be careful not to provide such a method if the ServiceTracker object is given to other bundles.

The services that are being tracked are available via a ServiceTracker. These services are dependent on the BundleContext as well. It is therefore necessary to do a careful security analysis when ServiceTracker objects are given to other bundles.

19.7 Changes

The only change in this specification has been the addition of the getTrackingCount method.

The implementation that is included with the interface sources has been partially rewritten to use less synchronization.

19.8 org.osgi.util.tracker

The OSGi Service Tracker 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.util.tracker; specification-version=1.2
```

19.8.1 Summary

- ServiceTracker – The ServiceTracker class simplifies using services from the Framework's service registry. [p.391]
- ServiceTrackerCustomizer – The ServiceTrackerCustomizer interface allows a ServiceTracker object to customize the service objects that are tracked. [p.391]

19.8.2 public class ServiceTracker implements ServiceTrackerCustomizer

The ServiceTracker class simplifies using services from the Framework's service registry.

A ServiceTracker object is constructed with search criteria and a ServiceTrackerCustomizer object. A ServiceTracker object can use the ServiceTrackerCustomizer object to customize the service objects to be tracked. The ServiceTracker object can then be opened to begin tracking all services in the Framework's service registry that match the specified search criteria. The ServiceTracker object correctly handles all of the details of listening to ServiceEvent objects and getting and ungetting services.

The getServiceReferences method can be called to get references to the services being tracked. The getService and getServices methods can be called to get the service objects for the tracked service.

19.8.2.1 protected final BundleContext context

Bundle context this ServiceTracker object is tracking against.

19.8.2.2 protected final Filter filter

Filter specifying search criteria for the services to track.

Since 1.1

19.8.2.3 public ServiceTracker(BundleContext context, ServiceReference reference, ServiceTrackerCustomizer customizer)

context BundleContext object against which the tracking is done.

reference ServiceReference object for the service to be tracked.

customizer The customizer object to call when services are added, modified, or removed in this ServiceTracker object. If customizer is null, then this ServiceTracker object will be used as the ServiceTrackerCustomizer object and the ServiceTracker object will call the ServiceTrackerCustomizer methods on itself.

- Create a ServiceTracker object on the specified ServiceReference object.

  The service referenced by the specified ServiceReference object will be tracked by this ServiceTracker object.

19.8.2.4 public ServiceTracker(BundleContext context, String clazz, ServiceTrackerCustomizer customizer)

context BundleContext object against which the tracking is done.

clazz Class name of the services to be tracked.

customizer The customizer object to call when services are added, modified, or removed in this ServiceTracker object. If customizer is null, then this ServiceTracker object will be used as the ServiceTrackerCustomizer object and the ServiceTracker object will call the ServiceTrackerCustomizer methods on itself.

- Create a ServiceTracker object on the specified class name.

Services registered under the specified class name will be tracked by this ServiceTracker object.

19.8.2.5 public ServiceTracker(BundleContext context, Filter filter, ServiceTrackerCustomizer customizer)

context BundleContext object against which the tracking is done.

filter Filter object to select the services to be tracked.

customizer The customizer object to call when services are added, modified, or removed in this ServiceTracker object. If customizer is null, then this ServiceTracker object will be used as the ServiceTrackerCustomizer object and the ServiceTracker object will call the ServiceTrackerCustomizer methods on itself.

- Create a ServiceTracker object on the specified Filter object.

  Services which match the specified Filter object will be tracked by this ServiceTracker object.

Since 1.1

19.8.2.6 public Object addingService(ServiceReference reference)

reference Reference to service being added to this ServiceTracker object.

- Default implementation of the ServiceTrackerCustomizer.addingService method.

  This method is only called when this ServiceTracker object has been constructed with a null ServiceTrackerCustomizer argument. The default implementation returns the result of calling getService, on the BundleContext object with which this ServiceTracker object was created, passing the specified ServiceReference object.

  This method can be overridden in a subclass to customize the service object to be tracked for the service being added. In that case, take care not to rely on the default implementation of removedService that will unget the service.

Returns The service object to be tracked for the service added to this ServiceTracker object.

See Also ServiceTrackerCustomizer[p.391]

19.8.2.7 public synchronized void close( )

- Close this ServiceTracker object.

  This method should be called when this ServiceTracker object should end the tracking of services.

19.8.2.8 protected void finalize( ) throws Throwable

- Properly close this ServiceTracker object when finalized. This method calls the close method to close this ServiceTracker object if it has not already been closed.

19.8.2.9 public Object getService(ServiceReference reference)

reference Reference to the desired service.

- Returns the service object for the specified `ServiceReference` object if the referenced service is being tracked by this `ServiceTracker` object.

Returns Service object or `null` if the service referenced by the specified `ServiceReference` object is not being tracked.

19.8.2.10 public Object getService( )

- Returns a service object for one of the services being tracked by this `ServiceTracker` object.

  If any services are being tracked, this method returns the result of calling `getService(getServiceReference())`.

Returns Service object or `null` if no service is being tracked.

19.8.2.11 public ServiceReference getServiceReference( )

- Returns a `ServiceReference` object for one of the services being tracked by this `ServiceTracker` object.

  If multiple services are being tracked, the service with the highest ranking (as specified in its `service.ranking` property) is returned.

  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 returned.

  This is the same algorithm used by `BundleContext.getServiceReference`.

Returns `ServiceReference` object or `null` if no service is being tracked.

Since 1.1

19.8.2.12 public ServiceReference[] getServiceReferences( )

- Return an array of `ServiceReference` objects for all services being tracked by this `ServiceTracker` object.

Returns Array of `ServiceReference` objects or `null` if no service are being tracked.

19.8.2.13 public Object[] getServices( )

- Return an array of service objects for all services being tracked by this `ServiceTracker` object.

Returns Array of service objects or `null` if no service are being tracked.

19.8.2.14 public int getTrackingCount( )

- Returns the tracking count for this `ServiceTracker` object. The tracking count is initialized to 0 when this `ServiceTracker` object is opened. Every time a service is added or removed from this `ServiceTracker` object the tracking count is incremented.

  The tracking count can be used to determine if this `ServiceTracker` object has added or removed a service by comparing a tracking count value previously collected with the current tracking count value. If the value has not changed, then no service has been added or removed from this `ServiceTracker` object since the previous tracking count was collected.

Returns The tracking count for this `ServiceTracker` object or -1 if this `ServiceTracker` object is not open.

Since 1.2

19.8.2.15 public void modifiedService(ServiceReference reference, Object service)

reference Reference to modified service.

service The service object for the modified service.

- Default implementation of the `ServiceTrackerCustomizer.modifiedService` method.

  This method is only called when this `ServiceTracker` object has been constructed with a `null ServiceTrackerCustomizer` argument. The default implementation does nothing.

See Also `ServiceTrackerCustomizer`[p.391]

19.8.2.16 public synchronized void open( )

- Open this `ServiceTracker` object and begin tracking services.

  Services which match the search criteria specified when this `ServiceTracker` object was created are now tracked by this `ServiceTracker` object.

Throws `IllegalStateException` – if the `BundleContext` object with which this `ServiceTracker` object was created is no longer valid.

19.8.2.17 public void remove(ServiceReference reference)

reference Reference to the service to be removed.

- Remove a service from this `ServiceTracker` object. The specified service will be removed from this `ServiceTracker` object. If the specified service was being tracked then the `ServiceTrackerCustomizer.removedService` method will be called for that service.

19.8.2.18 public void removedService(ServiceReference reference, Object object)

reference Reference to removed service.

service The service object for the removed service.

- Default implementation of the `ServiceTrackerCustomizer.removedService` method.

  This method is only called when this `ServiceTracker` object has been constructed with a `null ServiceTrackerCustomizer` argument. The default implementation calls `ungetService`, on the `BundleContext` object with which this `ServiceTracker` object was created, passing the specified `ServiceReference` object.

  This method can be overridden in a subclass. If the default implementation of `addingService` method was used, this method must unget the service.

See Also `ServiceTrackerCustomizer`[p.391]

19.8.2.19 public int size( )

- Return the number of services being tracked by this `ServiceTracker` object.

Returns Number of services being tracked.

19.8.2.20 public Object waitForService(long timeout) throws

InterruptedException

timeout time interval in milliseconds to wait. If zero, the method will wait indefinately.

- Wait for at least one service to be tracked by this `ServiceTracker` object.

  It is strongly recommended that `waitForService` is not used during the calling of the `BundleActivator` methods. `BundleActivator` methods are expected to complete in a short period of time.

Returns Returns the result of `getService()`.

Throws `IllegalArgumentException` – If the value of timeout is negative.

19.8.3 public interface ServiceTrackerCustomizer

The `ServiceTrackerCustomizer` interface allows a `ServiceTracker` object to customize the service objects that are tracked. The `ServiceTrackerCustomizer` object is called when a service is being added to the `ServiceTracker` object. The `ServiceTrackerCustomizer` can then return an object for the tracked service. The `ServiceTrackerCustomizer` object is also called when a tracked service is modified or has been removed from the `ServiceTracker` object.

The methods in this interface may be called as the result of a `ServiceEvent` being received by a `ServiceTracker` object. Since `ServiceEvents` are synchronously delivered by the Framework, it is highly recommended that implementations of these methods do not register (`BundleContext.registerService`), modify (`ServiceRegistration.setProperties`) or unregister (`ServiceRegistration.unregister`) a service while being synchronized on any object.

19.8.3.1 public Object addingService(ServiceReference reference)

reference Reference to service being added to the `ServiceTracker` object.

- A service is being added to the `ServiceTracker` object.

  This method is called before a service which matched the search parameters of the `ServiceTracker` object is added to it. This method should return the service object to be tracked for this `ServiceReference` object. The returned service object is stored in the `ServiceTracker` object and is available from the `getService` and `getServices` methods.

Returns The service object to be tracked for the `ServiceReference` object or `null` if the `ServiceReference` object should not be tracked.

19.8.3.2 public void modifiedService(ServiceReference reference, Object service)

reference Reference to service that has been modified.

service The service object for the modified service.

- A service tracked by the `ServiceTracker` object has been modified.

  This method is called when a service being tracked by the `ServiceTracker` object has had it properties modified.

19.8.3.3 public void removedService(ServiceReference reference, Object service)

reference Reference to service that has been removed.

service The service object for the removed service.

- A service tracked by the `ServiceTracker` object has been removed.

  This method is called after a service is no longer being tracked by the `ServiceTracker` object.

20 Measurement and State Specification

Version 1.0

20.1 Introduction

The Measurement class is a utility that provides a consistent way of handling a diverse range of measurements for bundle developers. Its purpose is to simplify the correct handling of measurements in OSGi Service Platforms.

OSGi bundle developers from all over the world have different preferences for measurement units, such as feet versus meters. In an OSGi environment, bundles developed in different parts of the world can and will exchange measurements when collaborating.

Distributing a measurement such as a simple floating point number requires the correct and equal understanding of the measurement's semantic by both the sender and the receiver. Numerous accidents have occurred due to misunderstandings between the sender and receiver because there are so many different ways to represent the same value. For example, on September 23, 1999, the Mars Polar Lander was lost because calculations used to program the craft's trajectory were input with English units while the operation documents specified metric units. See [62] *Mars Polar Lander failure* for more information.

This Measurement and State Specification defines the norm that should be used by all applications that execute in an OSGi Service Platform. This specification also provides utility classes.

20.1.1 Measurement Essentials

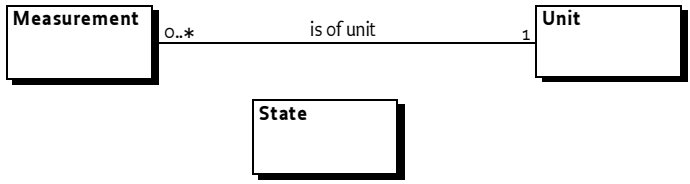
- *Numerical error* – All floating point measurements should be able to have a numerical error.
- *Numerical error calculations simplification* – Support should be provided to simplify measurements calculations.
- *Unit conflict resolution* – It must not be possible to perform addition or subtraction with different units when they are not compatible. For example, it must not be possible to add meters to amperes or watts to pascals.
- *Unit coercion* – Multiplication and division operations involving more than one type of measurement must result in a different unit. For example, if meters are divided by seconds, the result must be a new unit that represents m/s.
- *Time-stamp* – Measurements should contain a time-stamp so that bundles can determine the age of a particular measurement.

- *Support for floating and discrete values* – Both floating point values (64 bit Java double floats) and discrete measurements (32 bit Java int) should be supported.
- *Consistency* – The method of error calculation and handling of unit types should be consistent.
- *Presentation* – The format of measurements and specified units should be easy to read and understand.

20.1.2 Measurement Entities

- *Measurement object* – A Measurement object contains a double value, a double error, and a long time-stamp. It is associated with a Unit object that represents its *type*.
- *State object* – A State object contains a discrete measurement (int) with a time-stamp and a name.
- *Unit object* – A Unit object represents a unit such as meter, second, mol, or Pascal. A number of Unit objects are predefined and have common names. Other Unit objects are created as needed from the 7 basic Système International d'Unité (SI) units. Different units are *not* used when a conversion is sufficient. For example, the unit of a Measurement object for length is *always* meters. If the length is needed in feet, then the number of feet is calculated by multiplying the value of the Measurement object in meters with the necessary conversion factor.
- *Error* – When a measurement is taken, it is *never* accurate. This specification defines the error as the value that is added and subtracted to the value to produce an interval, where the probability is 95% that the actual value falls within this interval.
- *Unit* – A unit is the *type* of a measurement: meter, feet, liter, gallon etc.
- *Base Unit* – One of the 7 base units defined in the SI.
- *Derived SI unit* – A unit is a derived SI unit when it is a combination of exponentiated base units. For example, a volt (V) is a derived unit because it can be expressed as $(m^2 \times kg) / (s^3 \times A)$, where m, kg, s and A are all base units.
- *Quantitative derivation* – A unit is quantitatively derived when it is converted to one of the base units or derived units using a conversion formula. For example, kilometers (km) can be converted to meters (m), gallons can be converted to liters, or horsepower can be converted to watts.

Figure 64 *Class Diagram, org.osgi.util.measurement*



20.2 Measurement Object

A Measurement object contains a value, an error, and a time-stamp It is linked to a Unit object that describes the measurement unit in an SI Base Unit or Derived SI Unit.

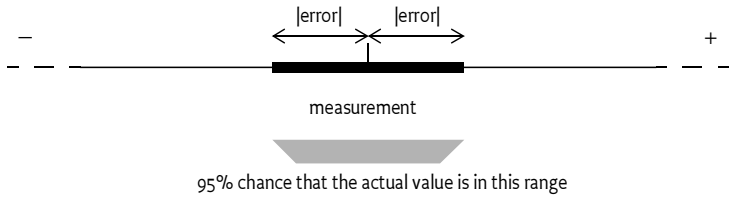
20.2.1 Value

The value of the Measurement object is the measured value. It is set in a constructor. The type of the value is double.

20.2.2 Error

The Measurement object can contain a numerical error. This error specifies an interval by adding and subtracting the error value from the measured value. The type of the error is double. A valid error value indicates that the actual measured value has a 95% chance of falling within this interval (see Figure 2). If the error is not known it should be represented as a Double.NaN.

Figure 65 The Error Interval



20.2.3 Time-stamp

When a Measurement object is created, the time-stamp can be set. A time-stamp is a long value representing the number of milliseconds since the epoch midnight of January 1, 1970, UTC (this is the value from System.currentTimeMillis() method).

By default, a time-stamp is not set because the call to System.currentTimeMillis() incurs overhead. If the time-stamp is not set when the Measurement object is created, then its default value is zero. If the time-stamp is set, the creator of the Measurement object must give the time as an argument to the constructor. For example:

```
Measurement m = new Measurement(
   v, e, null, System.currentTimeMillis() );
```

20.3 Error Calculations

Once a measurement is taken, it often is used in calculations. The error value assigned to the result of a calculation depends largely on the error values of the operands. Therefore, the Measurement class offers addition, subtraction, multiplication, and division functions for measurements and constants. These functions take the error into account when performing the specific operation.

The Measurement class uses absolute errors and has methods to calculate a new absolute error when multiplication, division, addition, or subtraction is performed. Error calculations must therefore adhere to the rules listed in Table 25. In this table, Δa is the absolute positive error in a value a and Δb is the absolute positive error in a value b. c is a constant floating point value without an error.

Calculation	Function	Error
$a \times b$	mul(Measurement)	$\lvert \Delta a \times b \rvert + \lvert a \times \Delta b \rvert$
a / b	div(Measurement)	$(\lvert \Delta a \times b \rvert + \lvert a \times \Delta b \rvert) / b^2$
$a + b$	add(Measurement)	$\Delta a + \Delta b$
$a - b$	sub(Measurement)	$\Delta a + \Delta b$
$a \times c$	mul(double)	$\lvert \Delta a \times c \rvert$
a / c	div(double)	$\lvert \Delta a / c \rvert$
$a + c$	add(double)	Δa
$a - c$	sub(double)	Δa

Table 25 Error Calculation Rules

20.4 Comparing Measurements

Measurement objects have a value and an error range, making comparing these objects more complicated than normal scalars.

20.4.1 Identity and Equality

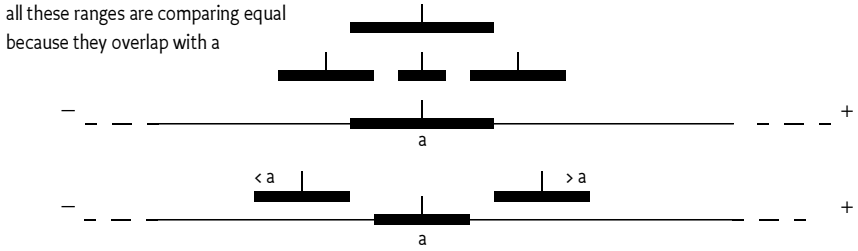
Both equals(Object) and hashCode() methods are overridden to provide value-based equality. Two Measurement objects are equal when the unit, error, and value are the same. The time-stamp is not relevant for equality or the hash code.

20.4.2 Comparing Measurement Objects

The Measurement class implements the java.lang.Comparable interface and thus implements the compareTo(Object) method. Comparing two Measurement objects is not straightforward, however, due to the associated error. The error effectively creates a range, so comparing two Measurement objects is actually comparing intervals.

Two Measurement objects are considered to be equal when their intervals overlap. In all other cases, the value is used in the comparison.

Figure 66 *Comparing Measurement Objects*



This comparison implies that the equals(Object) method may return false while the compareTo(Object) method returns 0 for the same Measurement object.

20.5 Unit Object

Each Measurement object is related to a Unit object. The Unit object defines the unit of the measurement value and error. For example, the Unit object might define the unit of the measurement value and the error as meters (m). For convenience, the Unit class defines a number of standard units as constants. Measurement objects are given a specific Unit with the constructor. The following example shows how a measurement can be associated with meters (m):

```
Measurement length = new Measurement( v, 0.01, Unit.m );
```

Units are based on the Système International d'Unité (SI), developed after the French Revolution. The SI consists of 7 different units that can be combined in many ways to form a large series of derived units. The basic 7 units are listed in Table 26. For more information, see [58] *General SI index*.

Description	Unit name	Symbol
length	meter	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
thermodynamic temperature	kelvin	K
amount of substance	mole	mol
luminous intensity	candela	cd

Table 26 *Basic SI units.*

Additional units are derived in the following ways:

Derived units can be a combination of exponentiated base units. For example, Hz (Hertz) is the unit for frequencies and is actually derived from the calculation of 1/s. A more complicated derived unit is volt (V). A volt is actually:

$$(m^2 \times kg) / (s^3 \times A)$$

The SI defines various derived units with their own name, for example pascal (Pa), watt (W), volt (V), and many more.

The Measurement class must maintain its unit by keeping track of the exponents of the 7 basic SI units.

If different units are used in addition or subtraction of Measurement objects, an ArithmeticException must be thrown.

```
Measurement length = new Measurement( v1, 0.01, Unit.m );
Measurement duration = new Measurement( v2, 0, Unit.s );
try {
   Measurement r = length.add( duration );
}
catch( ArithmeticException e ) {
   // This must be thrown
}
```

When two Measurement objects are multiplied, the Unit object of the result contains the sum of the exponents. When two Measurement objects are divided, the exponents of the Unit object of the result are calculated by subtraction of the exponents.

The Measurement class must support exponents of -64 to +63. Overflow must not be reported but must result in an invalid Unit object. All calculations with an invalid Unit object should result in an invalid Unit object. Typical computations generate exponents for units between +/- 4.

20.5.1 Quantitive Differences

The base and derived units can be converted to other units that are of the same *quality*, but require a conversion because their scales and offsets may differ. For example, degrees Fahrenheit, kelvin, and Celsius are all temperatures and, therefore, only differ in their quantity. Kelvin and Celsius are the same scale and differ only in their starting points. Fahrenheit differs from kelvin in that both scale and starting point differ.

Using different Unit objects for the units that differ only in quantity can easily introduce serious software bugs. Therefore, the Unit class utilizes the SI units. Any exchange of measurements should be done using SI units to prevent these errors. When a measurement needs to be displayed, the presentation logic should perform the necessary conversions to present it in a localized form. For example, when speed is presented in a car purchased in the United States, it should be presented as miles instead of meters.

20.5.2 Why Use SI Units?

The adoption of the SI in the United States and the United Kingdom has met with resistance. This issue raises the question why the SI system has to be the preferred measurement system in the OSGi Specifications.

The SI system is utilized because it is the only measurement *system* that has a consistent set of base units. The base units can be combined to create a large number of derived units without requiring a large number of complicated conversion formulas. For example, a watt is simply a combination of meters, kilograms, and seconds ($m^2 \times kg/s^3$). In contrast, horsepower is not easily related to inches, feet, fathoms, yards, furlongs, ounces, pounds, stones, or miles. This difficulty is the reason that science has utilized the SI for a long time. It is also the reason that the SI has been chosen as the system used for the Measurement class.

The purpose of the Measurement class is internal, however, and should not restrict the usability of the OSGi environment. Users should be able to use the local measurement units when data is input or displayed. This choice is the responsibility of the application developer.

20.6 State Object

The State object is used to represent discrete states. It contains a time-stamp but does not contain an error or Unit object. The Measurement object is not suitable to maintain discrete states. For example, a car door can be LOCKED, UNLOCKED, or CHILDLOCKED. Measuring and operating with these values does not require error calculations, nor does it require SI units. Therefore, the State object is a simple, named object that holds an integer value.

20.7 Related Standards

20.7.1 JSR 108 Units Specification

Sun Microsystems Java Community Process (JCP) [59] *JSR 108 Units Specification* addresses the same issues as this Measurement and State Specification. At the time of the writing of this specification, no public review of the JCP has occurred. This JSR, however, seems to be based on the [60] *Unidata user group: MetaApps project*. This Unidata API overlaps this specification but has the following issues:

- It uses a significant number of classes. Using many small classes can create significant overhead, which is a problem for the resource-constrained OSGi Service Platform.
- It treats the SI units in the same way as quantitatively derived units. As explained earlier, the purpose of the Measurement class is to prevent confusion between units that only differ in their quantity like gallons and liters. It is considered better to strictly separate the presentation of units from the units used in calculations.
- It is not yet complete as of the writing of this specification.

This JSR does not seem to move past the initial review phase.

20.7.2 GNU Math Library in Kawa

The open source project Kawa, a scheme-based Java environment, has included a gnu.math library that contains unit handling similar to this specification. It can be found at [61] *A Math Library containing unit handling in Kawa*.

The library seems considerably more complex without offering much more functionality than this specification. It also does not strictly separate basic SI units such as meter from quantitatively derived units such as pica.

20.8 Security Considerations

The Measurement, Unit, and State classes have been made immutable. Instances of these classes can be freely handed out to other bundles because they cannot be extended, nor can the value, error, or time-stamp be altered after the object is created.

20.9 org.osgi.util.measurement

The OSGi Measurement 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.util.measurement; specification-version=1.0
```

20.9.1 Summary

- Measurement – Represents a value with an error, a unit and a time-stamp. [p.410]
- State – Groups a state name, value and timestamp. [p.415]
- Unit – A unit system for measurements. [p.405]

20.9.2 public class Measurement implements Comparable

Represents a value with an error, a unit and a time-stamp.

A Measurement object is used for maintaining the tuple of value, error, unit and time-stamp. The value and error are represented as doubles and the time is measured in milliseconds since midnight, January 1, 1970 UTC.

Mathematic methods are provided that correctly calculate taking the error into account. A runtime error will occur when two measurements are used in an incompatible way. E.g., when a speed (m/s) is added to a distance (m). The measurement class will correctly track changes in unit during multiplication and division, always coercing the result to the most simple form. See Unit[p.405] for more information on the supported units.

Errors in the measurement class are absolute errors. Measurement errors should use the P95 rule. Actual values must fall in the range value +/- error 95% or more of the time.

A Measurement object is immutable in order to be easily shared.

Note: This class has a natural ordering that is inconsistent with equals. See compareTo[p.412].

20.9.2.1 public Measurement(double value, double error, Unit unit, long time)

value The value of the Measurement.

error The error of the Measurement.

unit The Unit object in which the value is measured. If this argument is null, then the unit will be set to Unit.unity[p.418].

time The time measured in milliseconds since midnight, January 1, 1970 UTC.

- Create a new Measurement object.

20.9.2.2 public Measurement(double value, double error, Unit unit)

value The value of the Measurement.

error The error of the Measurement.

unit The Unit object in which the value is measured. If this argument is null, then the unit will be set to Unit.unity[p.418].

- Create a new Measurement object with a time of zero.

20.9.2.3 public Measurement(double value, Unit unit)

value The value of the Measurement.

unit The Unit in which the value is measured. If this argument is null, then the unit will be set to Unit.unity[p.418].

- Create a new Measurement object with an error of 0.0 and a time of zero.

20.9.2.4 public Measurement(double value)

value The value of the Measurement.

- Create a new Measurement object with an error of 0.0, a unit of Unit.unity[p.418] and a time of zero.

20.9.2.5 public Measurement add(Measurement m)

m The Measurement object that will be added with this object.

- Returns a new Measurement object that is the sum of this object added to the specified object. The error and unit of the new object are computed. The time of the new object is set to the time of this object.

Returns A new Measurement object that is the sum of this and m.

Throws ArithmeticException – If the Unit objects of this object and the specified object cannot be added.

See Also Unit[p.405]

20.9.2.6 public Measurement add(double d, Unit u)

d The value that will be added with this object.

u The Unit object of the specified value.

- Returns a new Measurement object that is the sum of this object added to the specified value.

Returns A new Measurement object that is the sum of this object added to the specified value. The unit of the new object is computed. The error and time of the new object is set to the error and time of this object.

Throws ArithmeticException – If the Unit objects of this object and the specified value cannot be added.

See Also Unit[p.405]

20.9.2.7 public Measurement add(double d)

d The value that will be added with this object.

- Returns a new Measurement object that is the sum of this object added to the specified value.

Returns A new Measurement object that is the sum of this object added to the specified value. The error, unit, and time of the new object is set to the error, Unit and time of this object.

20.9.2.8 public int compareTo(Object obj)

obj The object to be compared.

- Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer if this object is less than, equal to, or greater than the specified object.

  Note: This class has a natural ordering that is inconsistent with equals. For this method, another Measurement object is considered equal if there is some x such that

  ```
  getValue()-getError() <= x<= getValue()+getError()
  ```

  for both Measurement objects being compared.

Returns A negative integer, zero, or a positive integer if this object is less than, equal to, or greater than the specified object.

Throws ClassCastException – If the specified object is not of type Measurement.

ArithmeticException – If the unit of the specified Measurement object is not equal to the Unit object of this object.

20.9.2.9 public Measurement div(Measurement m)

m The Measurement object that will be the divisor of this object.

- Returns a new Measurement object that is the quotient of this object divided by the specified object.

Returns A new Measurement object that is the quotient of this object divided by the specified object. The error and unit of the new object are computed. The time of the new object is set to the time of this object.

Throws ArithmeticException – If the Unit objects of this object and the specified object cannot be divided.

See Also Unit[p.405]

20.9.2.10 public Measurement div(double d, Unit u)

d The value that will be the divisor of this object.

u The `Unit` object of the specified value.

- Returns a new `Measurement` object that is the quotient of this object divided by the specified value.

Returns A new `Measurement` that is the quotient of this object divided by the specified value. The error and unit of the new object are computed. The time of the new object is set to the time of this object.

Throws `ArithmeticException` – If the `Unit` objects of this object and the specified object cannot be divided.

See Also `Unit`[p.405]

20.9.2.11 public Measurement div(double d)

d The value that will be the divisor of this object.

- Returns a new `Measurement` object that is the quotient of this object divided by the specified value.

Returns A new `Measurement` object that is the quotient of this object divided by the specified value. The error of the new object is computed. The unit and time of the new object is set to the `Unit` and time of this object.

20.9.2.12 public boolean equals(Object obj)

obj The object to compare with this object.

- Returns whether the specified object is equal to this object. Two `Measurement` objects are equal if they have same value, error and `Unit`.

  Note: This class has a natural ordering that is inconsistent with equals. See `compareTo`[p.412] .

Returns `true` if this object is equal to the specified object; `false` otherwise.

20.9.2.13 public final double getError( )

- Returns the error of this `Measurement` object. The error is always a positive value.

Returns The error of this `Measurement` as a double.

20.9.2.14 public final long getTime( )

- Returns the time at which this `Measurement` object was taken. The time is measured in milliseconds since midnight, January 1, 1970 UTC, or zero when not defined.

Returns The time at which this `Measurement` object was taken or zero.

20.9.2.15 public final Unit getUnit( )

- Returns the `Unit` object of this `Measurement` object.

Returns The `Unit` object of this `Measurement` object.

See Also `Unit`[p.405]

20.9.2.16 public final double getValue( )

- Returns the value of this `Measurement` object.

Returns The value of this Measurement object as a double.

20.9.2.17 public int hashCode( )

- Returns a hash code value for this object.

Returns A hash code value for this object.

20.9.2.18 public Measurement mul(Measurement m)

m The Measurement object that will be multiplied with this object.

- Returns a new Measurement object that is the product of this object multiplied by the specified object.

Returns A new Measurement that is the product of this object multiplied by the specified object. The error and unit of the new object are computed. The time of the new object is set to the time of this object.

Throws ArithmeticException – If the Unit objects of this object and the specified object cannot be multiplied.

See Also Unit[p.405]

20.9.2.19 public Measurement mul(double d, Unit u)

d The value that will be multiplied with this object.

u The Unit of the specified value.

- Returns a new Measurement object that is the product of this object multiplied by the specified value.

Returns A new Measurement object that is the product of this object multiplied by the specified value. The error and unit of the new object are computed. The time of the new object is set to the time of this object.

Throws ArithmeticException – If the units of this object and the specified value cannot be multiplied.

See Also Unit[p.405]

20.9.2.20 public Measurement mul(double d)

d The value that will be multiplied with this object.

- Returns a new Measurement object that is the product of this object multiplied by the specified value.

Returns A new Measurement object that is the product of this object multiplied by the specified value. The error of the new object is computed. The unit and time of the new object is set to the unit and time of this object.

20.9.2.21 public Measurement sub(Measurement m)

m The Measurement object that will be subtracted from this object.

- Returns a new Measurement object that is the subtraction of the specified object from this object.

Returns A new Measurement object that is the subtraction of the specified object from this object. The error and unit of the new object are computed. The time of the new object is set to the time of this object.

Throws ArithmeticException – If the Unit objects of this object and the specified object cannot be subtracted.

See Also Unit[p.405]

20.9.2.22 public Measurement sub(double d, Unit u)

d The value that will be subtracted from this object.

u The Unit object of the specified value.

□ Returns a new Measurement object that is the subtraction of the specified value from this object.

Returns A new Measurement object that is the subtraction of the specified value from this object. The unit of the new object is computed. The error and time of the new object is set to the error and time of this object.

Throws ArithmeticException – If the Unit objects of this object and the specified object cannot be subtracted.

See Also Unit[p.405]

20.9.2.23 public Measurement sub(double d)

d The value that will be subtracted from this object.

□ Returns a new Measurement object that is the subtraction of the specified value from this object.

Returns A new Measurement object that is the subtraction of the specified value from this object. The error, unit and time of the new object is set to the error, Unit object and time of this object.

20.9.2.24 public String toString( )

□ Returns a String object representing this Measurement object.

Returns a String object representing this Measurement object.

20.9.3 public class State

Groups a state name, value and timestamp.

The state itself is represented as an integer and the time is measured in milliseconds since midnight, January 1, 1970 UTC.

A State object is immutable so that it may be easily shared.

20.9.3.1 public State(int value, String name, long time)

value The value of the state.

name The name of the state.

time The time measured in milliseconds since midnight, January 1, 1970 UTC.

□ Create a new State object.

20.9.3.2 public State(int value, String name)

value The value of the state.

name The name of the state.

□ Create a new State object with a time of 0.

20.9.3.3 public boolean equals(Object obj)

obj The object to compare with this object.

- Return whether the specified object is equal to this object. Two State objects are equal if they have same value and name.

Returns true if this object is equal to the specified object; false otherwise.

20.9.3.4 public final String getName( )

- Returns the name of this State.

Returns The name of this State object.

20.9.3.5 public final long getTime( )

- Returns the time with which this State was created.

Returns The time with which this State was created. The time is measured in milliseconds since midnight, January 1, 1970 UTC.

20.9.3.6 public final int getValue( )

- Returns the value of this State.

Returns The value of this State object.

20.9.3.7 public int hashCode( )

- Returns a hash code value for this object.

Returns A hash code value for this object.

20.9.3.8 public String toString( )

- Returns a String object representing this object.

Returns a String object representing this object.

20.9.4 public class Unit

A unit system for measurements. This class contains definitions of the most common SI units.

This class only support exponents for the base SI units in the range -64 to +63. Any operation which produces an exponent outside of this range will result in a Unit object with undefined exponents.

20.9.4.1 public static final Unit A

The electric current unit ampere (A)

20.9.4.2 public static final Unit C

The electric charge unit coulomb (C).

coulomb is expressed in SI units as s· A

20.9.4.3 public static final Unit cd

The luminous intensity unit candela (cd)

20.9.4.4 public static final Unit F

The capacitance unit farad (F).

farad is equal to C/V or is expressed in SI units as s4· A2/m2· kg

20.9.4.5 public static final Unit Gy

The absorbed dose unit gray (Gy).

Gy is equal to J/kg or is expressed in SI units as m2/s2

20.9.4.6 public static final Unit Hz

The frequency unit hertz (Hz).

hertz is expressed in SI units as 1/s

20.9.4.7 public static final Unit J

The energy unit joule (J).

joule is equal to N· m or is expressed in SI units as m2· kg/s2

20.9.4.8 public static final Unit K

The temperature unit kelvin (K)

20.9.4.9 public static final Unit kat

The catalytic activity unit katal (kat).

katal is expressed in SI units as mol/s

20.9.4.10 public static final Unit kg

The mass unit kilogram (kg)

20.9.4.11 public static final Unit lx

The illuminance unit lux (lx).

lux is expressed in SI units as cd/m2

20.9.4.12 public static final Unit m

The length unit meter (m)

20.9.4.13 public static final Unit m2

The area unit square meter(m2)

20.9.4.14 public static final Unit m3

The volume unit cubic meter (m3)

20.9.4.15 public static final Unit m_s

The speed unit meter per second (m/s)

20.9.4.16 public static final Unit m_s2

The acceleration unit meter per second squared (m/s2)

20.9.4.17 public static final Unit mol

The amount of substance unit mole (mol)

20.9.4.18 **public static final Unit N**

The force unit newton (N).

N is expressed in SI units as m· kg/s2

20.9.4.19 **public static final Unit Ohm**

The electric resistance unit ohm.

ohm is equal to V/A or is expressed in SI units as m2· kg/s3· A2

20.9.4.20 **public static final Unit Pa**

The pressure unit pascal (Pa).

Pa is equal to N/m2 or is expressed in SI units as kg/m· s2

20.9.4.21 **public static final Unit rad**

The angle unit radians (rad)

20.9.4.22 **public static final Unit S**

The electric conductance unit siemens (S).

siemens is equal to A/V or is expressed in SI units as s3· A2/m2· kg

20.9.4.23 **public static final Unit s**

The time unit second (s)

20.9.4.24 **public static final Unit T**

The magnetic flux density unit tesla (T).

tesla is equal to Wb/m2 or is expressed in SI units as kg/s2· A

20.9.4.25 **public static final Unit unity**

No Unit (Unity)

20.9.4.26 **public static final Unit V**

The electric potential difference unit volt (V).

volt is equal to W/A or is expressed in SI units as m2· kg/s3· A

20.9.4.27 **public static final Unit W**

The power unit watt (W).

watt is equal to J/s or is expressed in SI units as m2· kg/s3

20.9.4.28 **public static final Unit Wb**

The magnetic flux unit weber (Wb).

weber is equal to V· s or is expressed in SI units as m2· kg/s2· A

20.9.4.29 **public boolean equals(Object obj)**

obj the `Unit` object that should be checked for equality

- Checks whether this Unit object is equal to the specified Unit object. The Unit objects are considered equal if their exponents are equal.

Returns true if the specified Unit object is equal to this Unit object.

20.9.4.30 public int hashCode( )

- Returns the hash code for this object.

Returns This object's hash code.

20.9.4.31 public String toString( )

- Returns a String object representing the Unit

Returns A String object representing the Unit

20.10 References

[57] *SI Units information*
http://physics.nist.gov/cuu/Units

[58] *General SI index*
http://directory.google.com/Top/Science/Reference/Units_of_Measurement

[59] *JSR 108 Units Specification*
http://www.jcp.org/jsr/detail/108.jsp

[60] *Unidata user group: MetaApps project*
http://www.unidata.ucar.edu/community/committees/metapps/docs/ucar/units/package-summary.html

[61] *A Math Library containing unit handling in Kawa*
http://www.gnu.org/software/kawa

[62] *Mars Polar Lander failure*
http://mars.jpl.nasa.gov/msp98/news/mco990930.html

21 Position Specification

Version 1.0

21.1 Introduction

The Position class is a utility providing bundle developers with a consistent way of handling geographic positions in OSGi applications. The Position class is intended to be used with the Wire Admin service but has wider applicability.

The Position class is designed to be compatible with the Global Positioning System (GPS). This specification will not define or explain the complexities of positioning information. It is assumed that the reader has the appropriate background to understand this information.

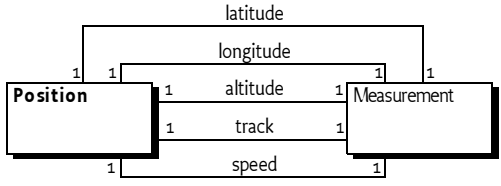
21.1.1 Essentials

- *Position* – Provide an information object that has well defined semantics for a position.
- *WGS-84* – Use the World Geodetic System 84 as the datum.
- *Speed* – Provide speed and track information.
- *Errors* – Position information always has certain errors or cannot be measured at all. This information must be available to the users of the information.
- *Units* – Use SI units for all measurements.
- *Wire Admin* – This specification must work within the Wire Admin service.

21.1.2 Entities

- *Position* – An object containing the different aspects of a position.
- *Measurement* – Contains a typed measurement made at a certain time and with a specified error.

Figure 67 *Class Diagram, org.osgi.util.position*



21.2 Positioning

The Position class is used to give information about the position and movement of a vehicle with a specified amount of uncertainty. The position is based on WGS-84.

The Position class offers the following information:

- getLatitude() – The WGS-84 latitude of the current position. The unit of a latitude must be rad (radians).
- getLongitude() – The WGS-84 longitude of the current position. The unit of a longitude must be rad (radians).
- getAltitude() – Altitude is expressed as height in meters above the WGS-84 ellipsoid. This value can differ from the actual height above mean sea level depending on the place on earth where the measurement is taken place. This value is not corrected for the geoid.
- getTrack() – The true north course of the vehicle in radians.
- getSpeed() – The ground speed. This speed must not include vertical speed.

21.3 Units

Longitude and latitude are represented in radians, not degrees. This is consistent with the use of the Measurement object. Radians can be converted to degrees with the following formula, when lonlat is the longitude or latitude:

```
degrees = (lonlat / π) * 180
```

Calculation errors are significantly reduced when all calculations are done with a single unit system. This approach increases the complexity of presentation, but presentations are usually localized and require conversion anyway. Also, the radians are the units in the SI system and the java.lang.Math class uses only radians for angles.

21.4 Optimizations

A Position object must be immutable. It must remain its original values after it is created.

The Position class is not final. This approach implies that developers are allowed to sub-class it and provide optimized implementations. For example, it is possible that the Measurement objects are only constructed when actually requested.

21.5 Errors

Positioning information is never exact. Even large errors can exist in certain conditions. For this reason, the Position class returns all its measurements as Measurement objects. The Measurement class maintains an error value for each measurement.

In certain cases it is not possible to supply a value; in those cases, the method should return a NaN as specified in the Measurement class.

21.6 Using Position With Wire Admin

The primary reason the Position is specified, is to use it with the *Wire Admin Service Specification* on page 325. A bundle that needs position information should register a Consumer service and the configuration should connect this service to an appropriate Producer service.

21.7 Related Standards

21.7.1 JSR 179

In JCP, started [65] *Location API for J2ME* . This API is targeted at embedded systems and is likely to not contain some of the features found in this API. This API is targeted to be reviewed at Q4 of 2002. This API should be considered in a following release.

21.8 Security

The security aspects of the Position class are delegated to the security aspects of the Wire Admin service. The Position object only carries the information. The Wire Admin service will define what Consumer services will receive position information from what Producer services. It is therefore up to the administrator of the Wire Admin service to assure that only trusted bundles receive this information, or can supply it.

21.9 org.osgi.util.position

The OSGi Position 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.util.position; specification-version=1.0
```

21.9.1 public class Position

Position represents a geographic location, based on the WGS84 System (World Geodetic System 1984).

The `org.osgi.util.measurement.Measurement` class is used to represent the values that make up a position.

A given position object may lack any of it's components, i.e. the altitude may not be known. Such missing values will be represented by null.

Position does not override the implementation of either equals() or hashCode() because it is not clear how missing values should be handled. It is up to the user of a position to determine how best to compare two position objects. A `Position` object is immutable.

21.9.1.1 public Position(Measurement lat, Measurement lon, Measurement alt, Measurement speed, Measurement track)

lat a `Measurement` object specifying the latitude in radians, or null

lon a `Measurement` object specifying the longitude in radians, or null

alt a `Measurement` object specifying the altitude in meters, or null

speed a `Measurement` object specifying the speed in meters per second, or null

track a `Measurement` object specifying the track in radians, or null

- Contructs a `Position` object with the given values.

21.9.1.2 public Measurement getAltitude( )

- Returns the altitude of this position in meters.

Returns a `Measurement` object in `Unit.m` representing the altitude in meters above the ellipsoid `null` if the altitude is not known.

21.9.1.3 public Measurement getLatitude( )

- Returns the latitude of this position in radians.

Returns a `Measurement` object in `Unit.rad` representing the latitude, or `null` if the latitude is not known..

21.9.1.4 public Measurement getLongitude( )

- Returns the longitude of this position in radians.

Returns a `Measurement` object in `Unit.rad` representing the longitude, or `null` if the longitude is not known.

21.9.1.5 public Measurement getSpeed( )

- Returns the ground speed of this position in meters per second.

Returns a `Measurement` object in `Unit.m_s` representing the speed, or `null` if the speed is not known..

21.9.1.6 public Measurement getTrack( )

- Returns the track of this position in radians as a compass heading. The track is the extrapolation of previous previously measured positions to a future position.

Returns a `Measurement` object in `Unit.rad` representing the track, or `null` if the track is not known..

21.10 References

[63] *World Geodetic System 84 (WGS-84)*
http://www.wgs84.com

[64] *Location Interoperability Forum*
http://www.locationforum.org/

[65] *Location API for J2ME*
http://www.jcp.org/jsr/detail/179.jsp

22 Execution Environment Specification

Version 1.0

22.1 Introduction

This specification defines two different execution environments for OSGi Server Platform Servers. One is based on a minimal environment that supports OSGi Framework and basic services implementations. The other is derived from [71] *Foundation Profile*. Care has been taken to make the minimum requirements a proper subset of Foundation Profile.

This chapter contains a detailed listing of the Execution Environments. This list is the actual specification and is normative. However, this list is not suited for tools. Therefore, the OSGi web site provides the JAR files that contain all the signatures of the Execution Environments on the OSGi web site, see [67] *Downloadable Execution Environments*.

Please note that the OSGi Minimum Execution Requirements do not constitute a specification for a Java technology profile or platform under the Java Community Process, but rather are a list of dependencies on certain elements of the presumed underlying Java profile(s) or platform(s).

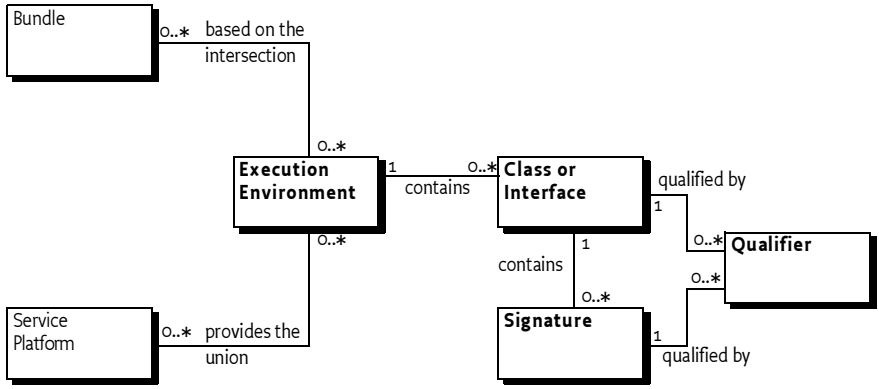
22.1.1 Essentials

- *Bundle Environment* – A well defined format with handling rules for defining the classes and methods that a bundle can rely on.
- *Machine Processable* – It should be easy to process the specification with tools to verify bundles and Service Platforms.
- *Standards* – It should be based on standards as much as possible. It must be compatible with [68] *J2ME, Java 2 Micro Edition*.

22.1.2 Entities

- *Execution Environment* – A collection of classes.
- *Class* – Contains a set of qualifiers and a set of signature for each method and field in that class.
- *Signature* – A unique identifier for the type associated with a field or the return type and argument types of a function.
- *Qualifiers* – A set of attributes that further define a signature.
- *Profile* – A SUN/JCP defined set of classes, based on a configuration.
- *Configuration* – A SUN/JCP defined set of classes and VM specification.

Figure 68 Entities involved in an Execution Environment



22.2 About Execution Environments

22.2.1 Signatures

An Execution Environment consists of a set of public and protected *signatures*. A signature is defined to be a unique identifier for a field or method with class and type information. For example, the signature of the wait(long) method in Object would be:

```
java/lang/Object.wait(J)V
```

The encoding of the signature is defined in [66] *The Java Virtual Machine Specification*.

For this specification, each signature includes a set of *qualifiers* that further qualify the field or method. These are the access qualifiers (like public, private, and protected), and informational qualifiers like synchronized, volatile, strictfp, interface, native, and abstract. These informational qualifiers are not included in the EE listings.

An Execution Environment consists of a set of classes and interfaces with their access qualifiers. Each class consist of a set of signatures.

22.2.2 Semantics

An Execution environment is solely based on the signatures of the methods and fields. An OSGi Execution Environment relies on the appropriate SUN Java documents to define the semantics of a methods or fields.

22.3 OSGi Defined Execution Environments

This specification contains two Execution Environments. They are listed in the following sections. Each signature is printed in the normal Java format except that public modifiers are not shown to save space (all fields or methods must be public or protected to be included in this list).

Before each signature there are two columns.

1. OSGi/Minimum-1.0 execution requirements
2. CDC-1.0/Foundation-1.0 execution environment.

If the column contains a ■, it means that the signature has been *included* in that Execution Environment. A □ indicates that the signature is missing from the EE.

The information is included here for completeness. However, it is likely that tools will be developed by vendors that validate the compliance of Service Platforms and bundles in relation to an Execution Environment. For that reason, it is possible to download a JAR file containing all the signatures as Java class files from the OSGi web site, see [67] *Downloadable Execution Environments*.

22.3.1 java.io

■ ■ **package java.io**

■ ■ **class BufferedInputStream extends FilterInputStream**

■ ■ BufferedInputStream(InputStream)
■ ■ BufferedInputStream(InputStream,int)
■ ■ int available() throws IOException
■ ■ protected byte[] buf
■ ■ void close() throws IOException
■ ■ protected int count
■ ■ void mark(int)
■ ■ protected int marklimit
■ ■ protected int markpos
■ ■ boolean markSupported()
■ ■ protected int pos
■ ■ int read() throws IOException
■ ■ int read(byte[],int,int) throws IOException
■ ■ void reset() throws IOException
■ ■ long skip(long) throws IOException

■ ■ **class BufferedOutputStream extends FilterOutputStream**

■ ■ BufferedOutputStream(OutputStream)
■ ■ BufferedOutputStream(OutputStream, int)
■ ■ protected byte[] buf
■ ■ protected int count
■ ■ void flush() throws IOException
■ ■ void write(byte[],int,int) throws IOException
■ ■ void write(int) throws IOException

■ ■ **class BufferedReader extends Reader**

■ ■ BufferedReader(Reader)
■ ■ BufferedReader(Reader,int)
■ ■ void close() throws IOException
■ ■ void mark(int) throws IOException
■ ■ boolean markSupported()
■ ■ int read() throws IOException
■ ■ int read(char[],int,int) throws IOException
■ ■ String readLine() throws IOException
■ ■ boolean ready() throws IOException
■ ■ void reset() throws IOException
■ ■ long skip(long) throws IOException

■ ■ **class BufferedWriter extends Writer**

■ ■ BufferedWriter(Writer)
■ ■ BufferedWriter(Writer,int)
■ ■ void close() throws IOException
■ ■ void flush() throws IOException
■ ■ void newLine() throws IOException
■ ■ void write(char[],int,int) throws IOException
■ ■ void write(int) throws IOException
■ ■ void write(String,int,int) throws IOException

■ ■ **class ByteArrayInputStream extends InputStream**

■ ■ ByteArrayInputStream(byte[])
■ ■ ByteArrayInputStream(byte[],int,int)
■ ■ int available()
■ ■ protected byte[] buf
■ ■ void close() throws IOException
■ ■ protected int count
■ ■ void mark(int)
■ ■ protected int mark
■ ■ boolean markSupported()
■ ■ protected int pos
■ ■ int read()
■ ■ int read(byte[],int,int)
■ ■ void reset()
■ ■ long skip(long)

■ ■ **class ByteArrayOutputStream extends OutputStream**

■ ■ ByteArrayOutputStream()
■ ■ ByteArrayOutputStream(int)
■ ■ protected byte[] buf
■ ■ void close() throws IOException
■ ■ protected int count
■ ■ void reset()
■ ■ int size()
■ ■ byte[] toByteArray()
■ ■ String toString()
■ ■ String toString(String) throws UnsupportedEncodingException
■ ■ void write(byte[],int,int)
■ ■ void write(int)
■ ■ void writeTo(OutputStream) throws IOException

□ ■ **class CharArrayReader extends Reader**

□ ■ CharArrayReader(char[])
□ ■ CharArrayReader(char[],int,int)

- protected char[] buf
- void close()
- protected int count
- void mark(int) throws IOException
- protected int markedPos
- boolean markSupported()
- protected int pos
- int read() throws IOException
- int read(char[],int,int) throws IOException
- boolean ready() throws IOException
- void reset() throws IOException
- long skip(long) throws IOException

class CharArrayWriter extends Writer

- CharArrayWriter()
- CharArrayWriter(int)
- protected char[] buf
- void close()
- protected int count
- void flush()
- void reset()
- int size()
- char[] toCharArray()
- String toString()
- void write(char[],int,int)
- void write(int)
- void write(String,int,int)
- void writeTo(Writer) throws IOException

class CharConversionException extends IOException

- CharConversionException()
- CharConversionException(String)

interface DataInput

- abstract boolean readBoolean() throws IOException
- abstract byte readByte() throws IOException
- abstract char readChar() throws IOException
- abstract double readDouble() throws IOException
- abstract float readFloat() throws IOException
- abstract void readFully(byte[]) throws IOException
- abstract void readFully(byte[],int,int) throws IOException
- abstract int readInt() throws IOException
- abstract String readLine() throws IOException
- abstract long readLong() throws IOException
- abstract short readShort() throws IOException
- abstract int readUnsignedByte() throws IOException
- abstract int readUnsignedShort() throws IOException
- abstract String readUTF() throws IOException
- abstract int skipBytes(int) throws IOException

class DataInputStream extends FilterInputStream implements DataInput

- DataInputStream(InputStream)
- final int read(byte[]) throws IOException
- final int read(byte[],int,int) throws IOException
- final boolean readBoolean() throws IOException
- final byte readByte() throws IOException
- final char readChar() throws IOException
- final double readDouble() throws IOException
- final float readFloat() throws IOException
- final void readFully(byte[]) throws IOException
- final void readFully(byte[],int,int) throws IOException
- final int readInt() throws IOException
- final String readLine() throws IOException
- final long readLong() throws IOException
- final short readShort() throws IOException
- final int readUnsignedByte() throws IOException
- final int readUnsignedShort() throws IOException
- final String readUTF() throws IOException
- final static String readUTF(DataInput) throws IOException
- final int skipBytes(int) throws IOException

interface DataOutput

- abstract void write(byte[]) throws IOException
- abstract void write(byte[],int,int) throws IOException
- abstract void write(int) throws IOException
- abstract void writeBoolean(boolean) throws IOException
- abstract void writeByte(int) throws IOException
- abstract void writeBytes(String) throws IOException
- abstract void writeChar(int) throws IOException
- abstract void writeChars(String) throws IOException
- abstract void writeDouble(double) throws IOException
- abstract void writeFloat(float) throws IOException
- abstract void writeInt(int) throws IOException
- abstract void writeLong(long) throws IOException
- abstract void writeShort(int) throws IOException
- abstract void writeUTF(String) throws IOException

class DataOutputStream extends FilterOutputStream implements DataOutput

- DataOutputStream(OutputStream)
- void flush() throws IOException
- final int size()
- void write(byte[],int,int) throws IOException
- void write(int) throws IOException
- final void writeBoolean(boolean) throws IOException
- final void writeByte(int) throws IOException

- final void writeBytes(String) throws IOException
- final void writeChar(int) throws IOException
- final void writeChars(String) throws IOException
- final void writeDouble(double) throws IOException
- final void writeFloat(float) throws IOException
- final void writeInt(int) throws IOException
- final void writeLong(long) throws IOException
- final void writeShort(int) throws IOException
- final void writeUTF(String) throws IOException
- protected int written

- **class EOFException extends IOException**
- EOFException()
- EOFException(String)

- **interface Externalizable implements Serializable**
- abstract void readExternal(ObjectInput) throws IOException, ClassNotFoundException
- abstract void writeExternal(ObjectOutput) throws IOException

- **class File implements Serializable , Comparable**
- File(File,String)
- File(String)
- File(String,String)
- boolean canRead()
- boolean canWrite()
- int compareTo(File)
- int compareTo(Object)
- boolean createNewFile() throws IOException
- static File createTempFile(String,String) throws IOException
- static File createTempFile(String,String, File) throws IOException
- boolean delete()
- void deleteOnExit()
- boolean equals(Object)
- boolean exists()
- File getAbsoluteFile()
- String getAbsolutePath()
- File getCanonicalFile() throws IOException
- String getCanonicalPath() throws IOException
- String getName()
- String getParent()
- File getParentFile()
- String getPath()
- int hashCode()
- boolean isAbsolute()
- boolean isDirectory()
- boolean isFile()
- boolean isHidden()
- long lastModified()
- long length()
- String[] list()
- String[] list(FilenameFilter)
- File[] listFiles()
- File[] listFiles(FileFilter)
- File[] listFiles(FilenameFilter)
- static File[] listRoots()
- boolean mkdir()
- boolean mkdirs()
- final static String pathSeparator
- final static char pathSeparatorChar
- boolean renameTo(File)
- final static String separator
- final static char separatorChar
- boolean setLastModified(long)
- boolean setReadOnly()
- String toString()
- java.net.URL toURL() throws java.net.MalformedURLException

- **final class FileDescriptor**
- FileDescriptor()
- final static FileDescriptor err
- final static FileDescriptor in
- final static FileDescriptor out
- void sync() throws SyncFailedException
- boolean valid()

- **interface FileFilter**
- abstract boolean accept(File)

- **class FileInputStream extends InputStream**
- FileInputStream(File) throws FileNotFoundException
- FileInputStream(FileDescriptor)
- FileInputStream(String) throws FileNotFoundException
- int available() throws IOException
- void close() throws IOException
- protected void finalize() throws IOException
- final FileDescriptor getFD() throws IOException
- int read() throws IOException
- int read(byte[]) throws IOException
- int read(byte[],int,int) throws IOException
- long skip(long) throws IOException

- **interface FilenameFilter**
- abstract boolean accept(File,String)

- **class FileNotFoundException extends IOException**
- FileNotFoundException()
- FileNotFoundException(String)

- **class FileOutputStream extends OutputStream**
- FileOutputStream(File) throws FileNotFoundException
- FileOutputStream(FileDescriptor)
- FileOutputStream(String) throws FileNotFoundException
- FileOutputStream(String,boolean) throws FileNotFoundException
- void close() throws IOException
- protected void finalize() throws IOException
- final FileDescriptor getFD() throws IOException
- void write(byte[]) throws IOException

- void write(byte[],int,int) throws IOException
- void write(int) throws IOException

- **final class FilePermission extends java.security.Permission implements Serializable**
- FilePermission(String,String)
- boolean equals(Object)
- String getActions()
- int hashCode()
- boolean implies(java.security.Permission)
- java.security.PermissionCollection newPermissionCollection()

- **class FileReader extends InputStreamReader**
- FileReader(File) throws FileNotFoundException
- FileReader(FileDescriptor)
- FileReader(String) throws FileNotFoundException

- **class FileWriter extends OutputStreamWriter**
- FileWriter(File) throws IOException
- FileWriter(FileDescriptor)
- FileWriter(String) throws IOException
- FileWriter(String,boolean) throws IOException

- **class FilterInputStream extends InputStream**
- protected FilterInputStream(InputStream)
- int available() throws IOException
- void close() throws IOException
- protected InputStream in
- void mark(int)
- boolean markSupported()
- int read() throws IOException
- int read(byte[]) throws IOException
- int read(byte[],int,int) throws IOException
- void reset() throws IOException
- long skip(long) throws IOException

- **class FilterOutputStream extends OutputStream**
- FilterOutputStream(OutputStream)
- void close() throws IOException
- void flush() throws IOException
- protected OutputStream out
- void write(byte[]) throws IOException
- void write(byte[],int,int) throws IOException
- void write(int) throws IOException

- **abstract class FilterReader extends Reader**
- protected FilterReader(Reader)
- void close() throws IOException
- protected Reader in
- void mark(int) throws IOException
- boolean markSupported()
- int read() throws IOException
- int read(char[],int,int) throws IOException
- boolean ready() throws IOException
- void reset() throws IOException
- long skip(long) throws IOException

- **abstract class FilterWriter extends Writer**
- protected FilterWriter(Writer)
- void close() throws IOException
- void flush() throws IOException
- protected Writer out
- void write(char[],int,int) throws IOException
- void write(int) throws IOException
- void write(String,int,int) throws IOException

- **abstract class InputStream**
- InputStream()
- int available() throws IOException
- void close() throws IOException
- void mark(int)
- boolean markSupported()
- abstract int read() throws IOException
- int read(byte[]) throws IOException
- int read(byte[],int,int) throws IOException
- void reset() throws IOException
- long skip(long) throws IOException

- **class InputStreamReader extends Reader**
- InputStreamReader(InputStream)
- InputStreamReader(InputStream,String) throws UnsupportedEncodingException
- void close() throws IOException
- String getEncoding()
- int read() throws IOException
- int read(char[],int,int) throws IOException
- boolean ready() throws IOException

- **class InterruptedIOException extends IOException**
- InterruptedIOException()
- InterruptedIOException(String)
- int bytesTransferred

- **class InvalidClassException extends ObjectStreamException**
- InvalidClassException(String)
- InvalidClassException(String,String)
- String classname
- String getMessage()

- **class InvalidObjectException extends ObjectStreamException**
- InvalidObjectException(String)

- **class IOException extends Exception**
- IOException()
- IOException(String)

- **class LineNumberReader extends BufferedReader**
- LineNumberReader(Reader)
- LineNumberReader(Reader,int)
- int getLineNumber()
- void mark(int) throws IOException
- int read() throws IOException
- int read(char[],int,int) throws IOException
- String readLine() throws IOException
- void reset() throws IOException
- void setLineNumber(int)

- long skip(long) throws IOException

class NotActiveException extends ObjectStreamException
- NotActiveException()
- NotActiveException(String)

class NotSerializableException extends ObjectStreamException
- NotSerializableException()
- NotSerializableException(String)

interface ObjectInput implements DataInput
- abstract int available() throws IOException
- abstract void close() throws IOException
- abstract int read() throws IOException
- abstract int read(byte[]) throws IOException
- abstract int read(byte[],int,int) throws IOException
- abstract Object readObject() throws ClassNotFoundException, IOException
- abstract long skip(long) throws IOException

class ObjectInputStream extends InputStream implements ObjectInput , ObjectStreamConstants
- protected ObjectInputStream() throws IOException, SecurityException
- ObjectInputStream(InputStream) throws StreamCorruptedException, IOException
- int available() throws IOException
- void close() throws IOException
- void defaultReadObject() throws IOException, ClassNotFoundException, NotActiveException
- protected boolean enableResolveObject(boolean) throws SecurityException
- int read() throws IOException
- int read(byte[],int,int) throws IOException
- boolean readBoolean() throws IOException
- byte readByte() throws IOException
- char readChar() throws IOException
- protected ObjectStreamClass readClassDescriptor() throws IOException, ClassNotFoundException
- double readDouble() throws IOException
- ObjectInputStream$GetField readFields() throws IOException, ClassNotFoundException, NotActiveException
- float readFloat() throws IOException
- void readFully(byte[]) throws IOException
- void readFully(byte[],int,int) throws IOException
- int readInt() throws IOException
- String readLine() throws IOException
- long readLong() throws IOException
- final Object readObject() throws OptionalDataException, ClassNotFoundException, IOException
- protected Object readObjectOverride() throws OptionalDataException, ClassNotFoundException, IOException
- short readShort() throws IOException
- protected void readStreamHeader() throws IOException, StreamCorruptedException
- int readUnsignedByte() throws IOException
- int readUnsignedShort() throws IOException
- String readUTF() throws IOException
- void registerValidation(ObjectInputValidation, int) throws NotActiveException, InvalidObjectException
- protected Class resolveClass(ObjectStreamClass) throws IOException, ClassNotFoundException
- protected Object resolveObject(Object) throws IOException
- protected Class resolveProxyClass(String[]) throws IOException, ClassNotFoundException
- int skipBytes(int) throws IOException

abstract class ObjectInputStream$GetField
- ObjectInputStream$GetField()
- abstract boolean defaulted(String) throws IOException, IllegalArgumentException
- abstract byte get(String,byte) throws IOException, IllegalArgumentException
- abstract char get(String,char) throws IOException, IllegalArgumentException
- abstract double get(String,double) throws IOException, IllegalArgumentException
- abstract float get(String,float) throws IOException, IllegalArgumentException
- abstract int get(String,int) throws IOException, IllegalArgumentException
- abstract long get(String,long) throws IOException, IllegalArgumentException
- abstract Object get(String,Object) throws IOException, IllegalArgumentException
- abstract short get(String,short) throws IOException, IllegalArgumentException
- abstract boolean get(String,boolean) throws IOException, IllegalArgumentException
- abstract ObjectStreamClass getObjectStreamClass()

interface ObjectInputValidation
- abstract void validateObject() throws InvalidObjectException

interface ObjectOutput implements DataOutput
- abstract void close() throws IOException
- abstract void flush() throws IOException
- abstract void write(byte[]) throws IOException
- abstract void write(byte[],int,int) throws IOException
- abstract void write(int) throws IOException
- abstract void writeObject(Object) throws IOException

class ObjectOutputStream extends OutputStream implements ObjectOutput ,

ObjectStreamConstants

- protected ObjectOutputStream() throws IOException, SecurityException
- ObjectOutputStream(OutputStream) throws IOException
- protected void annotateClass(Class) throws IOException
- protected void annotateProxyClass(Class) throws IOException
- void close() throws IOException
- void defaultWriteObject() throws IOException
- protected void drain() throws IOException
- protected boolean enableReplaceObject(boolean) throws SecurityException
- void flush() throws IOException
- ObjectOutputStream$PutField putFields() throws IOException
- protected Object replaceObject(Object) throws IOException
- void reset() throws IOException
- void useProtocolVersion(int) throws IOException
- void write(byte[]) throws IOException
- void write(byte[],int,int) throws IOException
- void write(int) throws IOException
- void writeBoolean(boolean) throws IOException
- void writeByte(int) throws IOException
- void writeBytes(String) throws IOException
- void writeChar(int) throws IOException
- void writeChars(String) throws IOException
- protected void writeClassDescriptor(ObjectStreamClass) throws IOException
- void writeDouble(double) throws IOException
- void writeFields() throws IOException
- void writeFloat(float) throws IOException
- void writeInt(int) throws IOException
- void writeLong(long) throws IOException
- final void writeObject(Object) throws IOException
- protected void writeObjectOverride(Object) throws IOException
- void writeShort(int) throws IOException
- protected void writeStreamHeader() throws IOException
- void writeUTF(String) throws IOException

- **abstract class ObjectOutputStream$PutField**
  - ObjectOutputStream$PutField()
  - abstract void put(String,byte)
  - abstract void put(String,char)
  - abstract void put(String,double)
  - abstract void put(String,float)
  - abstract void put(String,int)
  - abstract void put(String,long)
  - abstract void put(String,Object)
  - abstract void put(String,short)
  - abstract void put(String,boolean)
  - abstract void write(ObjectOutput) throws IOException

- **class ObjectStreamClass implements Serializable**
  - Class forClass()
  - ObjectStreamField getField(String)
  - ObjectStreamField[] getFields()
  - String getName()
  - long getSerialVersionUID()
  - static ObjectStreamClass lookup(Class)
  - final static ObjectStreamField[] NO_FIELDS
  - String toString()

- **interface ObjectStreamConstants**
  - final static int baseWireHandle
  - final static int PROTOCOL_VERSION_1
  - final static int PROTOCOL_VERSION_2
  - final static byte SC_BLOCK_DATA
  - final static byte SC_EXTERNALIZABLE
  - final static byte SC_SERIALIZABLE
  - final static byte SC_WRITE_METHOD
  - final static short STREAM_MAGIC
  - final static short STREAM_VERSION
  - final static SerializablePermission SUBCLASS_IMPLEMENTATION_PERMISSION
  - final static SerializablePermission SUBSTITUTION_PERMISSION
  - final static byte TC_ARRAY
  - final static byte TC_BASE
  - final static byte TC_BLOCKDATA
  - final static byte TC_BLOCKDATALONG
  - final static byte TC_CLASS
  - final static byte TC_CLASSDESC
  - final static byte TC_ENDBLOCKDATA
  - final static byte TC_EXCEPTION
  - final static byte TC_LONGSTRING
  - final static byte TC_MAX
  - final static byte TC_NULL
  - final static byte TC_OBJECT
  - final static byte TC_PROXYCLASSDESC
  - final static byte TC_REFERENCE
  - final static byte TC_RESET
  - final static byte TC_STRING

- **abstract class ObjectStreamException extends IOException**
  - protected ObjectStreamException()
  - protected ObjectStreamException(String)

- **class ObjectStreamField implements Comparable**
  - ObjectStreamField(String,Class)
  - int compareTo(Object)
  - String getName()
  - int getOffset()
  - Class getType()
  - char getTypeCode()
  - String getTypeString()
  - boolean isPrimitive()
  - protected void setOffset(int)
  - String toString()

- **class OptionalDataException extends ObjectStreamException**
  - boolean eof
  - int length

- **abstract class OutputStream**
  - OutputStream()
  - void close() throws IOException
  - void flush() throws IOException
  - void write(byte[]) throws IOException

- void write(byte[],int,int) throws IOException
- abstract void write(int) throws IOException

class OutputStreamWriter extends Writer

- OutputStreamWriter(OutputStream)
- OutputStreamWriter(OutputStream, String) throws UnsupportedEncodingException
- void close() throws IOException
- void flush() throws IOException
- String getEncoding()
- void write(char[],int,int) throws IOException
- void write(int) throws IOException
- void write(String,int,int) throws IOException

class PipedInputStream extends InputStream

- PipedInputStream()
- PipedInputStream(PipedOutputStream) throws IOException
- int available() throws IOException
- protected byte[] buffer
- void close() throws IOException
- void connect(PipedOutputStream) throws IOException
- protected int in
- protected int out
- final protected static int PIPE_SIZE
- int read() throws IOException
- int read(byte[],int,int) throws IOException
- protected void receive(int) throws IOException

class PipedOutputStream extends OutputStream

- PipedOutputStream()
- PipedOutputStream(PipedInputStream) throws IOException
- void close() throws IOException
- void connect(PipedInputStream) throws IOException
- void flush() throws IOException
- void write(byte[],int,int) throws IOException
- void write(int) throws IOException

class PipedReader extends Reader

- PipedReader()
- PipedReader(PipedWriter) throws IOException
- void close() throws IOException
- void connect(PipedWriter) throws IOException
- int read() throws IOException
- int read(char[],int,int) throws IOException
- boolean ready() throws IOException

class PipedWriter extends Writer

- PipedWriter()
- PipedWriter(PipedReader) throws IOException
- void close() throws IOException
- void connect(PipedReader) throws IOException
- void flush() throws IOException
- void write(char[],int,int) throws IOException
- void write(int) throws IOException

class PrintStream extends FilterOutputStream

- PrintStream(OutputStream)
- PrintStream(OutputStream,boolean)
- boolean checkError()
- void close()
- void flush()
- void print(char[])
- void print(char)
- void print(double)
- void print(float)
- void print(int)
- void print(long)
- void print(Object)
- void print(String)
- void print(boolean)
- void println()
- void println(char[])
- void println(char)
- void println(double)
- void println(float)
- void println(int)
- void println(long)
- void println(Object)
- void println(String)
- void println(boolean)
- protected void setError()
- void write(byte[],int,int)
- void write(int)

class PrintWriter extends Writer

- PrintWriter(OutputStream)
- PrintWriter(OutputStream,boolean)
- PrintWriter(Writer)
- PrintWriter(Writer,boolean)
- boolean checkError()
- void close()
- void flush()
- protected Writer out
- void print(char[])
- void print(char)
- void print(double)
- void print(float)
- void print(int)
- void print(long)
- void print(Object)
- void print(String)
- void print(boolean)
- void println()
- void println(char[])
- void println(char)
- void println(double)
- void println(float)
- void println(int)
- void println(long)
- void println(Object)
- void println(String)
- void println(boolean)
- protected void setError()
- void write(char[])
- void write(char[],int,int)
- void write(int)
- void write(String)

- void write(String,int,int)

- **class PushbackInputStream extends FilterInputStream**
  - PushbackInputStream(InputStream)
  - PushbackInputStream(InputStream,int)
  - int available() throws IOException
  - protected byte[] buf
  - void close() throws IOException
  - boolean markSupported()
  - protected int pos
  - int read() throws IOException
  - int read(byte[],int,int) throws IOException
  - long skip(long) throws IOException
  - void unread(byte[]) throws IOException
  - void unread(byte[],int,int) throws IOException
  - void unread(int) throws IOException

- **class PushbackReader extends FilterReader**
  - PushbackReader(Reader)
  - PushbackReader(Reader,int)
  - void close() throws IOException
  - void mark(int) throws IOException
  - boolean markSupported()
  - int read() throws IOException
  - int read(char[],int,int) throws IOException
  - boolean ready() throws IOException
  - void reset() throws IOException
  - void unread(char[]) throws IOException
  - void unread(char[],int,int) throws IOException
  - void unread(int) throws IOException

- **class RandomAccessFile implements DataInput , DataOutput**
  - RandomAccessFile(File,String) throws FileNotFoundException
  - RandomAccessFile(String,String) throws FileNotFoundException
  - void close() throws IOException
  - final FileDescriptor getFD() throws IOException
  - long getFilePointer() throws IOException
  - long length() throws IOException
  - int read() throws IOException
  - int read(byte[]) throws IOException
  - int read(byte[],int,int) throws IOException
  - final boolean readBoolean() throws IOException
  - final byte readByte() throws IOException
  - final char readChar() throws IOException
  - final double readDouble() throws IOException
  - final float readFloat() throws IOException
  - final void readFully(byte[]) throws IOException
  - final void readFully(byte[],int,int) throws IOException
  - final int readInt() throws IOException
  - final String readLine() throws IOException
  - final long readLong() throws IOException
  - final short readShort() throws IOException
  - final int readUnsignedByte() throws IOException
  - final int readUnsignedShort() throws IOException
  - final String readUTF() throws IOException
  - void seek(long) throws IOException
  - void setLength(long) throws IOException
  - int skipBytes(int) throws IOException
  - void write(byte[]) throws IOException
  - void write(byte[],int,int) throws IOException
  - void write(int) throws IOException
  - final void writeBoolean(boolean) throws IOException
  - final void writeByte(int) throws IOException
  - final void writeBytes(String) throws IOException
  - final void writeChar(int) throws IOException
  - final void writeChars(String) throws IOException
  - final void writeDouble(double) throws IOException
  - final void writeFloat(float) throws IOException
  - final void writeInt(int) throws IOException
  - final void writeLong(long) throws IOException
  - final void writeShort(int) throws IOException
  - final void writeUTF(String) throws IOException

- **abstract class Reader**
  - protected Reader()
  - protected Reader(Object)
  - abstract void close() throws IOException
  - protected Object lock
  - void mark(int) throws IOException
  - boolean markSupported()
  - int read() throws IOException
  - int read(char[]) throws IOException
  - abstract int read(char[],int,int) throws IOException
  - boolean ready() throws IOException
  - void reset() throws IOException
  - long skip(long) throws IOException

- **class SequenceInputStream extends InputStream**
  - SequenceInputStream(InputStream, InputStream)
  - SequenceInputStream(java.util.Enumeration)
  - int available() throws IOException
  - void close() throws IOException
  - int read() throws IOException
  - int read(byte[],int,int) throws IOException

- **interface Serializable**

- **final class SerializablePermission extends java.security.BasicPermission**
  - SerializablePermission(String)
  - SerializablePermission(String,String)

- **class StreamCorruptedException extends ObjectStreamException**

- StreamCorruptedException()
- StreamCorruptedException(String)

- **class StreamTokenizer**
- StreamTokenizer(Reader)
- void commentChar(int)
- void eolIsSignificant(boolean)
- int lineno()
- void lowerCaseMode(boolean)
- int nextToken() throws IOException
- double nval
- void ordinaryChar(int)
- void ordinaryChars(int,int)
- void parseNumbers()
- void pushBack()
- void quoteChar(int)
- void resetSyntax()
- void slashSlashComments(boolean)
- void slashStarComments(boolean)
- String sval
- String toString()
- final static int TT_EOF
- final static int TT_EOL
- final static int TT_NUMBER
- final static int TT_WORD
- int ttype
- void whitespaceChars(int,int)
- void wordChars(int,int)

- **class StringReader extends Reader**
- StringReader(String)
- void close()
- void mark(int) throws IOException
- boolean markSupported()
- int read() throws IOException
- int read(char[],int,int) throws IOException
- boolean ready() throws IOException
- void reset() throws IOException
- long skip(long) throws IOException

- **class StringWriter extends Writer**
- StringWriter()
- StringWriter(int)
- void close() throws IOException
- void flush()
- StringBuffer getBuffer()
- String toString()
- void write(char[],int,int)
- void write(int)
- void write(String)
- void write(String,int,int)

- **class SyncFailedException extends IOException**
- SyncFailedException(String)

- **class UnsupportedEncodingException extends IOException**
- UnsupportedEncodingException()
- UnsupportedEncodingException(String)

- **class UTFDataFormatException extends IOException**
- UTFDataFormatException()
- UTFDataFormatException(String)

- **class WriteAbortedException extends ObjectStreamException**
- WriteAbortedException(String,Exception)
- Exception detail
- String getMessage()

- **abstract class Writer**
- protected Writer()
- protected Writer(Object)
- abstract void close() throws IOException
- abstract void flush() throws IOException
- protected Object lock
- void write(char[]) throws IOException
- abstract void write(char[],int,int) throws IOException
- void write(int) throws IOException
- void write(String) throws IOException
- void write(String,int,int) throws IOException

22.3.2 java.lang

- **package java.lang**

- **class AbstractMethodError extends IncompatibleClassChangeError**
- AbstractMethodError()
- AbstractMethodError(String)

- **class ArithmeticException extends RuntimeException**
- ArithmeticException()
- ArithmeticException(String)

- **class ArrayIndexOutOfBoundsException extends IndexOutOfBoundsException**
- ArrayIndexOutOfBoundsException()
- ArrayIndexOutOfBoundsException(int)
- ArrayIndexOutOfBoundsException(String)

- **class ArrayStoreException extends RuntimeException**
- ArrayStoreException()
- ArrayStoreException(String)

- **final class Boolean implements java.io.Serializable**
- Boolean(String)
- Boolean(boolean)
- boolean booleanValue()
- boolean equals(Object)
- final static Boolean FALSE
- static boolean getBoolean(String)
- int hashCode()
- String toString()
- final static Boolean TRUE
- final static Class TYPE
- static Boolean valueOf(String)

- **final class Byte extends Number implements Comparable**
- Byte(byte)
- Byte(String) throws NumberFormatException
- byte byteValue()
- int compareTo(Byte)
- int compareTo(Object)
- static Byte decode(String) throws NumberFormatException
- double doubleValue()
- boolean equals(Object)
- float floatValue()
- int hashCode()
- int intValue()

- long longValue()
- final static byte MAX_VALUE
- final static byte MIN_VALUE
- static byte parseByte(String) throws NumberFormatException
- static byte parseByte(String,int) throws NumberFormatException
- short shortValue()
- String toString()
- static String toString(byte)
- final static Class TYPE
- static Byte valueOf(String) throws NumberFormatException
- static Byte valueOf(String,int) throws NumberFormatException

final class Character implements java.io.Serializable , Comparable

- Character(char)
- char charValue()
- final static byte COMBINING_SPACING_MARK
- int compareTo(Character)
- int compareTo(Object)
- final static byte CONNECTOR_PUNCTUATION
- final static byte CONTROL
- final static byte CURRENCY_SYMBOL
- final static byte DASH_PUNCTUATION
- final static byte DECIMAL_DIGIT_NUMBER
- static int digit(char,int)
- final static byte ENCLOSING_MARK
- final static byte END_PUNCTUATION
- boolean equals(Object)
- static char forDigit(int,int)
- final static byte FORMAT
- static int getNumericValue(char)
- static int getType(char)
- int hashCode()
- static boolean isDefined(char)
- static boolean isDigit(char)
- static boolean isIdentifierIgnorable(char)
- static boolean isISOControl(char)
- static boolean isJavaIdentifierPart(char)
- static boolean isJavaIdentifierStart(char)
- static boolean isLetter(char)
- static boolean isLetterOrDigit(char)
- static boolean isLowerCase(char)
- static boolean isSpaceChar(char)
- static boolean isTitleCase(char)
- static boolean isUnicodeIdentifierPart(char)
- static boolean isUnicodeIdentifierStart(char)
- static boolean isUpperCase(char)
- static boolean isWhitespace(char)
- final static byte LETTER_NUMBER
- final static byte LINE_SEPARATOR
- final static byte LOWERCASE_LETTER
- final static byte MATH_SYMBOL
- final static int MAX_RADIX
- final static char MAX_VALUE
- final static int MIN_RADIX
- final static char MIN_VALUE
- final static byte MODIFIER_LETTER
- final static byte MODIFIER_SYMBOL
- final static byte NON_SPACING_MARK
- final static byte OTHER_LETTER
- final static byte OTHER_NUMBER
- final static byte OTHER_PUNCTUATION
- final static byte OTHER_SYMBOL
- final static byte PARAGRAPH_SEPARATOR
- final static byte PRIVATE_USE
- final static byte SPACE_SEPARATOR
- final static byte START_PUNCTUATION
- final static byte SURROGATE
- final static byte TITLECASE_LETTER
- static char toLowerCase(char)
- String toString()
- static char toTitleCase(char)
- static char toUpperCase(char)
- final static Class TYPE
- final static byte UNASSIGNED
- final static byte UPPERCASE_LETTER

class Character$Subset

- protected Character$Subset(String)
- final boolean equals(Object)
- final int hashCode()
- final String toString()

final class Character$UnicodeBlock extends Character$Subset

- final static Character$UnicodeBlock ALPHABETIC_PRESENTATION_FORMS
- final static Character$UnicodeBlock ARABIC
- final static Character$UnicodeBlock ARABIC_PRESENTATION_FORMS_A
- final static Character$UnicodeBlock ARABIC_PRESENTATION_FORMS_B
- final static Character$UnicodeBlock ARMENIAN
- final static Character$UnicodeBlock ARROWS
- final static Character$UnicodeBlock BASIC_LATIN
- final static Character$UnicodeBlock BENGALI
- final static Character$UnicodeBlock BLOCK_ELEMENTS
- final static Character$UnicodeBlock BOPOMOFO
- final static Character$UnicodeBlock BOX_DRAWING
- final static Character$UnicodeBlock CJK_COMPATIBILITY
- final static Character$UnicodeBlock CJK_COMPATIBILITY_FORMS
- final static Character$UnicodeBlock CJK_COMPATIBILITY_IDEOGRAPHS
- final static Character$UnicodeBlock CJK_SYMBOLS_AND_PUNCTUATION
- final static Character$UnicodeBlock CJK_UNIFIED_IDEOGRAPHS
- final static Character$UnicodeBlock COMBINING_DIACRITICAL_MARKS
- final static Character$UnicodeBlock COMBINING_HALF_MARKS
- final static Character$UnicodeBlock COMBINING_MARKS_FOR_SYMBOLS
- final static Character$UnicodeBlock CONTROL_PICTURES
- final static Character$UnicodeBlock CURRENCY_SYMBOLS
- final static Character$UnicodeBlock CYRILLIC
- final static Character$UnicodeBlock DEVANAGARI
- final static Character$UnicodeBlock DINGBATS

- final static Character$UnicodeBlock ENCLOSED_ALPHANUMERICS
- final static Character$UnicodeBlock ENCLOSED_CJK_LETTERS_AND_MONTHS
- final static Character$UnicodeBlock GENERAL_PUNCTUATION
- final static Character$UnicodeBlock GEOMETRIC_SHAPES
- final static Character$UnicodeBlock GEORGIAN
- final static Character$UnicodeBlock GREEK
- final static Character$UnicodeBlock GREEK_EXTENDED
- final static Character$UnicodeBlock GUJARATI
- final static Character$UnicodeBlock GURMUKHI
- final static Character$UnicodeBlock HALFWIDTH_AND_FULLWIDTH_FORMS
- final static Character$UnicodeBlock HANGUL_COMPATIBILITY_JAMO
- final static Character$UnicodeBlock HANGUL_JAMO
- final static Character$UnicodeBlock HANGUL_SYLLABLES
- final static Character$UnicodeBlock HEBREW
- final static Character$UnicodeBlock HIRAGANA
- final static Character$UnicodeBlock IPA_EXTENSIONS
- final static Character$UnicodeBlock KANBUN
- final static Character$UnicodeBlock KANNADA
- final static Character$UnicodeBlock KATAKANA
- final static Character$UnicodeBlock LAO
- final static Character$UnicodeBlock LATIN_1_SUPPLEMENT
- final static Character$UnicodeBlock LATIN_EXTENDED_A
- final static Character$UnicodeBlock LATIN_EXTENDED_ADDITIONAL
- final static Character$UnicodeBlock LATIN_EXTENDED_B
- final static Character$UnicodeBlock LETTERLIKE_SYMBOLS
- final static Character$UnicodeBlock MALAYALAM
- final static Character$UnicodeBlock MATHEMATICAL_OPERATORS
- final static Character$UnicodeBlock MISCELLANEOUS_SYMBOLS
- final static Character$UnicodeBlock MISCELLANEOUS_TECHNICAL
- final static Character$UnicodeBlock NUMBER_FORMS
- static Character$UnicodeBlock of(char)
- final static Character$UnicodeBlock OPTICAL_CHARACTER_RECOGNITION
- final static Character$UnicodeBlock ORIYA
- final static Character$UnicodeBlock PRIVATE_USE_AREA
- final static Character$UnicodeBlock SMALL_FORM_VARIANTS
- final static Character$UnicodeBlock SPACING_MODIFIER_LETTERS
- final static Character$UnicodeBlock SPECIALS
- final static Character$UnicodeBlock SUPERSCRIPTS_AND_SUBSCRIPTS
- final static Character$UnicodeBlock SURROGATES_AREA
- final static Character$UnicodeBlock TAMIL
- final static Character$UnicodeBlock TELUGU
- final static Character$UnicodeBlock THAI
- final static Character$UnicodeBlock TIBETAN

final class Class implements java.io.Serializable

- static Class forName(String) throws ClassNotFoundException
- static Class forName(String,boolean, ClassLoader) throws ClassNotFoundException
- Class[] getClasses()
- ClassLoader getClassLoader()
- Class getComponentType()
- Constructor getConstructor(Class[]) throws NoSuchMethodException, SecurityException
- Constructor[] getConstructors() throws SecurityException
- Class[] getDeclaredClasses() throws SecurityException
- Constructor getDeclaredConstructor(Class[]) throws NoSuchMethodException, SecurityException
- Constructor[] getDeclaredConstructors() throws SecurityException
- Field getDeclaredField(String) throws NoSuchFieldException, SecurityException
- Field[] getDeclaredFields() throws SecurityException
- Method getDeclaredMethod(String, Class[]) throws NoSuchMethodException, SecurityException
- Method[] getDeclaredMethods() throws SecurityException
- Class getDeclaringClass()
- Field getField(String) throws NoSuchFieldException, SecurityException
- Field[] getFields() throws SecurityException
- Class[] getInterfaces()
- Method getMethod(String,Class[]) throws NoSuchMethodException, SecurityException
- Method[] getMethods() throws SecurityException
- int getModifiers()
- String getName()
- Package getPackage()
- java.security.ProtectionDomain getProtectionDomain()
- java.net.URL getResource(String)
- java.io.InputStream getResourceAsStream(String)
- Object[] getSigners()
- Class getSuperclass()
- boolean isArray()
- boolean isAssignableFrom(Class)
- boolean isInstance(Object)
- boolean isInterface()
- boolean isPrimitive()

- Object newInstance() throws IllegalAccessException, InstantiationException
- String toString()

class ClassCastException extends RuntimeException
- ClassCastException()
- ClassCastException(String)

class ClassCircularityError extends LinkageError
- ClassCircularityError()
- ClassCircularityError(String)

class ClassFormatError extends LinkageError
- ClassFormatError()
- ClassFormatError(String)

abstract class ClassLoader
- protected ClassLoader()
- protected ClassLoader(ClassLoader)
- final protected Class defineClass(String, byte[],int,int) throws ClassFormatError
- final protected Class defineClass(String, byte[],int,int, java.security.ProtectionDomain) throws ClassFormatError
- protected Package definePackage(String, String,String,String,String,String, java.net.URL) throws IllegalArgumentException
- protected Class findClass(String) throws ClassNotFoundException
- protected String findLibrary(String)
- final protected Class findLoadedClass(String)
- protected java.net.URL findResource(String)
- protected java.util.Enumeration findResources(String) throws java.io.IOException
- final protected Class findSystemClass(String) throws ClassNotFoundException
- protected Package getPackage(String)
- protected Package[] getPackages()
- final ClassLoader getParent()
- java.net.URL getResource(String)
- java.io.InputStream getResourceAsStream(String)
- final java.util.Enumeration getResources(String) throws java.io.IOException
- static ClassLoader getSystemClassLoader()
- static java.net.URL getSystemResource(String)
- static java.io.InputStream getSystemResourceAsStream(String)
- static java.util.Enumeration getSystemResources(String) throws java.io.IOException
- Class loadClass(String) throws ClassNotFoundException
- protected Class loadClass(String, boolean) throws ClassNotFoundException
- final protected void resolveClass(Class)
- final protected void setSigners(Class, Object[])

class ClassNotFoundException extends Exception
- ClassNotFoundException()
- ClassNotFoundException(String)
- ClassNotFoundException(String, Throwable)
- Throwable getException()
- void printStackTrace()
- void printStackTrace(java.io.PrintStream)
- void printStackTrace(java.io.PrintWriter)

interface Cloneable

class CloneNotSupportedException extends Exception
- CloneNotSupportedException()
- CloneNotSupportedException(String)

interface Comparable
- abstract int compareTo(Object)

final class Compiler
- static Object command(Object)
- static boolean compileClass(Class)
- static boolean compileClasses(String)
- static void disable()
- static void enable()

final class Double extends Number implements Comparable
- Double(double)
- Double(String) throws NumberFormatException
- byte byteValue()
- int compareTo(Double)
- int compareTo(Object)
- static long doubleToLongBits(double)
- static long doubleToRawLongBits(double)
- double doubleValue()
- boolean equals(Object)
- float floatValue()
- int hashCode()
- int intValue()
- boolean isInfinite()
- static boolean isInfinite(double)
- boolean isNaN()
- static boolean isNaN(double)
- static double longBitsToDouble(long)
- long longValue()
- final static double MAX_VALUE
- final static double MIN_VALUE
- final static double NaN
- final static double NEGATIVE_INFINITY
- static double parseDouble(String) throws NumberFormatException
- final static double POSITIVE_INFINITY
- short shortValue()
- String toString()
- static String toString(double)
- final static Class TYPE
- static Double valueOf(String) throws NumberFormatException

class Error extends Throwable
- Error()
- Error(String)

class Exception extends Throwable
- Exception()
- Exception(String)

class ExceptionInInitializerError extends LinkageError
- ExceptionInInitializerError()
- ExceptionInInitializerError(String)
- ExceptionInInitializerError(Throwable)
- Throwable getException()
- void printStackTrace()
- void printStackTrace(java.io.PrintStream)
- void printStackTrace(java.io.PrintWriter)

final class Float extends Number implements Comparable
- Float(double)
- Float(float)
- Float(String) throws NumberFormatException
- byte byteValue()
- int compareTo(Float)
- int compareTo(Object)
- double doubleValue()
- boolean equals(Object)
- static int floatToIntBits(float)
- static int floatToRawIntBits(float)
- float floatValue()
- int hashCode()
- static float intBitsToFloat(int)
- int intValue()
- boolean isInfinite()
- static boolean isInfinite(float)
- boolean isNaN()
- static boolean isNaN(float)
- long longValue()
- final static float MAX_VALUE
- final static float MIN_VALUE
- final static float NaN
- final static float NEGATIVE_INFINITY
- static float parseFloat(String) throws NumberFormatException
- final static float POSITIVE_INFINITY
- short shortValue()
- String toString()
- static String toString(float)
- final static Class TYPE
- static Float valueOf(String) throws NumberFormatException

class IllegalAccessError extends IncompatibleClassChangeError
- IllegalAccessError()
- IllegalAccessError(String)

class IllegalAccessException extends Exception
- IllegalAccessException()
- IllegalAccessException(String)

class IllegalArgumentException extends RuntimeException
- IllegalArgumentException()
- IllegalArgumentException(String)

class IllegalMonitorStateException extends RuntimeException
- IllegalMonitorStateException()
- IllegalMonitorStateException(String)

class IllegalStateException extends RuntimeException
- IllegalStateException()
- IllegalStateException(String)

class IllegalThreadStateException extends IllegalArgumentException
- IllegalThreadStateException()
- IllegalThreadStateException(String)

class IncompatibleClassChangeError extends LinkageError
- IncompatibleClassChangeError()
- IncompatibleClassChangeError(String)

class IndexOutOfBoundsException extends RuntimeException
- IndexOutOfBoundsException()
- IndexOutOfBoundsException(String)

class InheritableThreadLocal extends ThreadLocal
- InheritableThreadLocal()
- protected Object childValue(Object)
- Object get()
- void set(Object)

class InstantiationError extends IncompatibleClassChangeError
- InstantiationError()
- InstantiationError(String)

class InstantiationException extends Exception
- InstantiationException()
- InstantiationException(String)

final class Integer extends Number implements Comparable
- Integer(int)
- Integer(String) throws NumberFormatException
- byte byteValue()
- int compareTo(Integer)
- int compareTo(Object)
- static Integer decode(String) throws NumberFormatException
- double doubleValue()
- boolean equals(Object)
- float floatValue()
- static Integer getInteger(String)
- static Integer getInteger(String,int)
- static Integer getInteger(String,Integer)
- int hashCode()
- int intValue()
- long longValue()
- final static int MAX_VALUE
- final static int MIN_VALUE
- static int parseInt(String) throws NumberFormatException
- static int parseInt(String,int) throws NumberFormatException
- short shortValue()
- static String toBinaryString(int)
- static String toHexString(int)
- static String toOctalString(int)
- String toString()
- static String toString(int)
- static String toString(int,int)
- final static Class TYPE
- static Integer valueOf(String) throws NumberFormatException
- static Integer valueOf(String,int) throws NumberFormatException

class InternalError extends VirtualMachineError
- InternalError()
- InternalError(String)

class InterruptedException extends Exception
- InterruptedException()
- InterruptedException(String)

class LinkageError extends Error

- LinkageError()
- LinkageError(String)

final class Long extends Number implements Comparable

- Long(long)
- Long(String) throws NumberFormatException
- byte byteValue()
- int compareTo(Long)
- int compareTo(Object)
- static Long decode(String) throws NumberFormatException
- double doubleValue()
- boolean equals(Object)
- float floatValue()
- static Long getLong(String)
- static Long getLong(String,long)
- static Long getLong(String,Long)
- int hashCode()
- int intValue()
- long longValue()
- final static long MAX_VALUE
- final static long MIN_VALUE
- static long parseLong(String) throws NumberFormatException
- static long parseLong(String,int) throws NumberFormatException
- short shortValue()
- static String toBinaryString(long)
- static String toHexString(long)
- static String toOctalString(long)
- String toString()
- static String toString(long)
- static String toString(long,int)
- final static Class TYPE
- static Long valueOf(String) throws NumberFormatException
- static Long valueOf(String,int) throws NumberFormatException

final class Math

- static double abs(double)
- static float abs(float)
- static int abs(int)
- static long abs(long)
- static double acos(double)
- static double asin(double)
- static double atan(double)
- static double atan2(double,double)
- static double ceil(double)
- static double cos(double)
- final static double E
- static double exp(double)
- static double floor(double)
- static double IEEEremainder(double, double)
- static double log(double)
- static double max(double,double)
- static float max(float,float)
- static int max(int,int)
- static long max(long,long)
- static double min(double,double)
- static float min(float,float)
- static int min(int,int)
- static long min(long,long)
- final static double PI
- static double pow(double,double)
- static double random()
- static double rint(double)
- static long round(double)
- static int round(float)
- static double sin(double)
- static double sqrt(double)
- static double tan(double)
- static double toDegrees(double)
- static double toRadians(double)

class NegativeArraySizeException extends RuntimeException

- NegativeArraySizeException()
- NegativeArraySizeException(String)

class NoClassDefFoundError extends LinkageError

- NoClassDefFoundError()
- NoClassDefFoundError(String)

class NoSuchFieldError extends IncompatibleClassChangeError

- NoSuchFieldError()
- NoSuchFieldError(String)

class NoSuchFieldException extends Exception

- NoSuchFieldException()
- NoSuchFieldException(String)

class NoSuchMethodError extends IncompatibleClassChangeError

- NoSuchMethodError()
- NoSuchMethodError(String)

class NoSuchMethodException extends Exception

- NoSuchMethodException()
- NoSuchMethodException(String)

class NullPointerException extends RuntimeException

- NullPointerException()
- NullPointerException(String)

abstract class Number implements java.io.Serializable

- Number()
- byte byteValue()
- abstract double doubleValue()
- abstract float floatValue()
- abstract int intValue()
- abstract long longValue()
- short shortValue()

class NumberFormatException extends IllegalArgumentException

- NumberFormatException()
- NumberFormatException(String)

class Object

- Object()
- protected Object clone() throws CloneNotSupportedException
- boolean equals(Object)
- protected void finalize() throws Throwable
- final Class getClass()
- int hashCode()
- final void notify()
- final void notifyAll()
- String toString()
- final void wait() throws InterruptedException
- final void wait(long) throws InterruptedException
- final void wait(long,int) throws InterruptedException

class OutOfMemoryError extends VirtualMachineError
- OutOfMemoryError()
- OutOfMemoryError(String)

class Package
- String getImplementationTitle()
- String getImplementationVendor()
- String getImplementationVersion()
- String getName()
- static Package getPackage(String)
- static Package[] getPackages()
- String getSpecificationTitle()
- String getSpecificationVendor()
- String getSpecificationVersion()
- int hashCode()
- boolean isCompatibleWith(String) throws NumberFormatException
- boolean isSealed()
- boolean isSealed(java.net.URL)
- String toString()

abstract class Process
- Process()
- abstract void destroy()
- abstract int exitValue()
- abstract java.io.InputStream getErrorStream()
- abstract java.io.InputStream getInputStream()
- abstract java.io.OutputStream getOutputStream()
- abstract int waitFor() throws InterruptedException

interface Runnable
- abstract void run()

class Runtime
- void addShutdownHook(Thread)
- Process exec(String[]) throws java.io.IOException
- Process exec(String[],String[]) throws java.io.IOException
- Process exec(String[],String[], java.io.File) throws java.io.IOException
- Process exec(String) throws java.io.IOException
- Process exec(String,String[]) throws java.io.IOException
- Process exec(String,String[],java.io.File) throws java.io.IOException
- void exit(int)
- long freeMemory()
- void gc()
- static Runtime getRuntime()
- void halt(int)
- void load(String)
- void loadLibrary(String)
- boolean removeShutdownHook(Thread)
- void runFinalization()
- long totalMemory()
- void traceInstructions(boolean)
- void traceMethodCalls(boolean)

class RuntimeException extends Exception
- RuntimeException()
- RuntimeException(String)

final class RuntimePermission extends java.security.BasicPermission
- RuntimePermission(String)
- RuntimePermission(String,String)

class SecurityException extends RuntimeException
- SecurityException()
- SecurityException(String)

class SecurityManager
- SecurityManager()
- void checkAccept(String,int)
- void checkAccess(Thread)
- void checkAccess(ThreadGroup)
- void checkAwtEventQueueAccess()
- void checkConnect(String,int)
- void checkConnect(String,int,Object)
- void checkCreateClassLoader()
- void checkDelete(String)
- void checkExec(String)
- void checkExit(int)
- void checkLink(String)
- void checkListen(int)
- void checkMemberAccess(Class,int)
- void checkMulticast(java.net.InetAddress)
- void checkMulticast(java.net.InetAddress, byte)
- void checkPackageAccess(String)
- void checkPackageDefinition(String)
- void checkPermission(java.security.Permission)
- void checkPermission(java.security.Permission,Object)
- void checkPrintJobAccess()
- void checkPropertiesAccess()
- void checkPropertyAccess(String)
- void checkRead(java.io.FileDescriptor)
- void checkRead(String)
- void checkRead(String,Object)
- void checkSecurityAccess(String)
- void checkSetFactory()
- void checkSystemClipboardAccess()
- boolean checkTopLevelWindow(Object)
- void checkWrite(java.io.FileDescriptor)
- void checkWrite(String)
- protected Class[] getClassContext()
- Object getSecurityContext()
- ThreadGroup getThreadGroup()

final class Short extends Number implements Comparable
- Short(String) throws NumberFormatException
- Short(short)
- byte byteValue()
- int compareTo(Object)
- int compareTo(Short)
- static Short decode(String) throws NumberFormatException
- double doubleValue()
- boolean equals(Object)
- float floatValue()
- int hashCode()
- int intValue()
- long longValue()
- final static short MAX_VALUE
- final static short MIN_VALUE

- static short parseShort(String) throws NumberFormatException
- static short parseShort(String,int) throws NumberFormatException
- short shortValue()
- String toString()
- static String toString(short)
- final static Class TYPE
- static Short valueOf(String) throws NumberFormatException
- static Short valueOf(String,int) throws NumberFormatException

class StackOverflowError extends VirtualMachineError

- StackOverflowError()
- StackOverflowError(String)

final class StrictMath

- static double abs(double)
- static float abs(float)
- static int abs(int)
- static long abs(long)
- static double acos(double)
- static double asin(double)
- static double atan(double)
- static double atan2(double,double)
- static double ceil(double)
- static double cos(double)
- final static double E
- static double exp(double)
- static double floor(double)
- static double IEEEremainder(double, double)
- static double log(double)
- static double max(double,double)
- static float max(float,float)
- static int max(int,int)
- static long max(long,long)
- static double min(double,double)
- static float min(float,float)
- static int min(int,int)
- static long min(long,long)
- final static double PI
- static double pow(double,double)
- static double random()
- static double rint(double)
- static long round(double)
- static int round(float)
- static double sin(double)
- static double sqrt(double)
- static double tan(double)
- static double toDegrees(double)
- static double toRadians(double)

final class String implements java.io.Serializable , Comparable

- String()
- String(byte[])
- String(byte[],int,int)
- String(byte[],int,int,String) throws java.io.UnsupportedEncodingException
- String(byte[],String) throws java.io.UnsupportedEncodingException
- String(char[])
- String(char[],int,int)
- String(String)
- String(StringBuffer)
- final static java.util.Comparator CASE_INSENSITIVE_ORDER
- char charAt(int)
- int compareTo(Object)
- int compareTo(String)
- int compareToIgnoreCase(String)
- String concat(String)
- static String copyValueOf(char[])
- static String copyValueOf(char[],int,int)
- boolean endsWith(String)
- boolean equals(Object)
- boolean equalsIgnoreCase(String)
- byte[] getBytes()
- byte[] getBytes(String) throws java.io.UnsupportedEncodingException
- void getChars(int,int,char[],int)
- int hashCode()
- int indexOf(int)
- int indexOf(int,int)
- int indexOf(String)
- int indexOf(String,int)
- String intern()
- int lastIndexOf(int)
- int lastIndexOf(int,int)
- int lastIndexOf(String)
- int lastIndexOf(String,int)
- int length()
- boolean regionMatches(int,String,int,int)
- boolean regionMatches(boolean,int, String,int,int)
- String replace(char,char)
- boolean startsWith(String)
- boolean startsWith(String,int)
- String substring(int)
- String substring(int,int)
- char[] toCharArray()
- String toLowerCase()
- String toLowerCase(java.util.Locale)
- String toString()
- String toUpperCase()
- String toUpperCase(java.util.Locale)
- String trim()
- static String valueOf(char[])
- static String valueOf(char[],int,int)
- static String valueOf(char)
- static String valueOf(double)
- static String valueOf(float)
- static String valueOf(int)
- static String valueOf(long)
- static String valueOf(Object)
- static String valueOf(boolean)

final class StringBuffer implements java.io.Serializable

- StringBuffer()
- StringBuffer(int)
- StringBuffer(String)
- StringBuffer append(char[])
- StringBuffer append(char[],int,int)
- StringBuffer append(char)
- StringBuffer append(double)
- StringBuffer append(float)
- StringBuffer append(int)
- StringBuffer append(long)
- StringBuffer append(Object)
- StringBuffer append(String)
- StringBuffer append(boolean)
- int capacity()
- char charAt(int)
- StringBuffer delete(int,int)
- StringBuffer deleteCharAt(int)
- void ensureCapacity(int)
- void getChars(int,int,char[],int)
- StringBuffer insert(int,char[])
- StringBuffer insert(int,char[],int,int)
- StringBuffer insert(int,char)

- StringBuffer insert(int,double)
- StringBuffer insert(int,float)
- StringBuffer insert(int,int)
- StringBuffer insert(int,long)
- StringBuffer insert(int,Object)
- StringBuffer insert(int,String)
- StringBuffer insert(int,boolean)
- int length()
- StringBuffer replace(int,int,String)
- StringBuffer reverse()
- void setCharAt(int,char)
- void setLength(int)
- String substring(int)
- String substring(int,int)
- String toString()

class StringIndexOutOfBoundsException extends IndexOutOfBoundsException

- StringIndexOutOfBoundsException()
- StringIndexOutOfBoundsException(int)
- StringIndexOutOfBoundsException(String)

final class System

- static void arraycopy(Object,int,Object, int,int)
- static long currentTimeMillis()
- final static java.io.PrintStream err
- static void exit(int)
- static void gc()
- static java.util.Properties getProperties()
- static String getProperty(String)
- static String getProperty(String,String)
- static SecurityManager getSecurityManager()
- static int identityHashCode(Object)
- final static java.io.InputStream in
- static void load(String)
- static void loadLibrary(String)
- static String mapLibraryName(String)
- final static java.io.PrintStream out
- static void runFinalization()
- static void setErr(java.io.PrintStream)
- static void setIn(java.io.InputStream)
- static void setOut(java.io.PrintStream)
- static void setProperties(java.util.Properties)
- static String setProperty(String,String)
- static void setSecurityManager(SecurityManager)

class Thread implements Runnable

- Thread()
- Thread(Runnable)
- Thread(Runnable,String)
- Thread(String)
- Thread(ThreadGroup,Runnable)
- Thread(ThreadGroup,Runnable,String)
- Thread(ThreadGroup,String)
- static int activeCount()
- final void checkAccess()
- static Thread currentThread()
- void destroy()
- static void dumpStack()
- static int enumerate(Thread[])
- ClassLoader getContextClassLoader()
- final String getName()
- final int getPriority()
- final ThreadGroup getThreadGroup()
- void interrupt()
- static boolean interrupted()
- final boolean isAlive()
- final boolean isDaemon()
- boolean isInterrupted()
- final void join() throws InterruptedException
- final void join(long) throws InterruptedException
- final void join(long,int) throws InterruptedException
- final static int MAX_PRIORITY
- final static int MIN_PRIORITY
- final static int NORM_PRIORITY
- void run()
- void setContextClassLoader(ClassLoader)
- final void setDaemon(boolean)
- final void setName(String)
- final void setPriority(int)
- static void sleep(long) throws InterruptedException
- static void sleep(long,int) throws InterruptedException
- void start()
- String toString()
- static void yield()

class ThreadDeath extends Error

- ThreadDeath()

class ThreadGroup

- ThreadGroup(String)
- ThreadGroup(ThreadGroup,String)
- int activeCount()
- int activeGroupCount()
- final void checkAccess()
- final void destroy()
- int enumerate(Thread[])
- int enumerate(Thread[],boolean)
- int enumerate(ThreadGroup[])
- int enumerate(ThreadGroup[],boolean)
- final int getMaxPriority()
- final String getName()
- final ThreadGroup getParent()
- final void interrupt()
- final boolean isDaemon()
- boolean isDestroyed()
- void list()
- final boolean parentOf(ThreadGroup)
- final void setDaemon(boolean)
- final void setMaxPriority(int)
- String toString()
- void uncaughtException(Thread, Throwable)

class ThreadLocal

- ThreadLocal()
- Object get()
- protected Object initialValue()
- void set(Object)

class Throwable implements java.io.Serializable

- Throwable()
- Throwable(String)
- Throwable fillInStackTrace()
- String getLocalizedMessage()
- String getMessage()
- void printStackTrace()

- void printStackTrace(java.io.PrintStream)
- void printStackTrace(java.io.PrintWriter)
- String toString()

- **class UnknownError extends VirtualMachineError**
- UnknownError()
- UnknownError(String)

- **class UnsatisfiedLinkError extends LinkageError**
- UnsatisfiedLinkError()
- UnsatisfiedLinkError(String)

- **class UnsupportedClassVersionError extends ClassFormatError**
- UnsupportedClassVersionError()
- UnsupportedClassVersionError(String)

- **class UnsupportedOperationException extends RuntimeException**
- UnsupportedOperationException()
- UnsupportedOperationException(String)

- **class VerifyError extends LinkageError**
- VerifyError()
- VerifyError(String)

- **abstract class VirtualMachineError extends Error**
- VirtualMachineError()
- VirtualMachineError(String)

- **final class Void**
- final static Class TYPE

22.3.3 java.lang.ref

- **package java.lang.ref**

- **class PhantomReference extends Reference**
- PhantomReference(Object, ReferenceQueue)
- Object get()

- **abstract class Reference**
- void clear()
- Object get()
- boolean enqueue()
- boolean isEnqueued()

- **class ReferenceQueue**
- ReferenceQueue()
- Reference poll()
- Reference remove() throws InterruptedException
- Reference remove(long) throws IllegalArgumentException, InterruptedException

- **class SoftReference extends Reference**
- SoftReference(Object)
- SoftReference(Object,ReferenceQueue)
- Object get()

- **class WeakReference extends Reference**
- WeakReference(Object)
- WeakReference(Object,ReferenceQueue)

22.3.4 java.lang.reflect

- **package java.lang.reflect**

- **class AccessibleObject**
- protected AccessibleObject()
- boolean isAccessible()
- static void setAccessible(AccessibleObject[], boolean) throws SecurityException
- void setAccessible(boolean) throws SecurityException

- **final class Array**
- static Object get(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static boolean getBoolean(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static byte getByte(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static char getChar(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static double getDouble(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static float getFloat(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static int getInt(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static int getLength(Object) throws IllegalArgumentException
- static long getLong(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static short getShort(Object,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static Object newInstance(Class,int[]) throws NegativeArraySizeException, IllegalArgumentException
- static Object newInstance(Class,int) throws NegativeArraySizeException
- static void set(Object,int,Object) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static void setBoolean(Object,int, boolean) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static void setByte(Object,int,byte) throws IllegalArgumentException, ArrayIndexOutOfBoundsException

- static void setChar(Object,int,char) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static void setDouble(Object,int,double) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static void setFloat(Object,int,float) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static void setInt(Object,int,int) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static void setLong(Object,int,long) throws IllegalArgumentException, ArrayIndexOutOfBoundsException
- static void setShort(Object,int,short) throws IllegalArgumentException, ArrayIndexOutOfBoundsException

final class Constructor extends AccessibleObject implements Member

- boolean equals(Object)
- Class getDeclaringClass()
- Class[] getExceptionTypes()
- int getModifiers()
- String getName()
- Class[] getParameterTypes()
- int hashCode()
- Object newInstance(Object[]) throws InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException
- String toString()

final class Field extends AccessibleObject implements Member

- boolean equals(Object)
- Object get(Object) throws IllegalAccessException, IllegalArgumentException
- boolean getBoolean(Object) throws IllegalAccessException, IllegalArgumentException
- byte getByte(Object) throws IllegalAccessException, IllegalArgumentException
- char getChar(Object) throws IllegalAccessException, IllegalArgumentException
- Class getDeclaringClass()
- double getDouble(Object) throws IllegalAccessException, IllegalArgumentException
- float getFloat(Object) throws IllegalAccessException, IllegalArgumentException
- int getInt(Object) throws IllegalAccessException, IllegalArgumentException
- long getLong(Object) throws IllegalAccessException, IllegalArgumentException
- int getModifiers()
- String getName()
- short getShort(Object) throws IllegalAccessException, IllegalArgumentException
- Class getType()
- int hashCode()
- void set(Object,Object) throws IllegalAccessException, IllegalArgumentException
- void setBoolean(Object,boolean) throws IllegalAccessException, IllegalArgumentException
- void setByte(Object,byte) throws IllegalAccessException, IllegalArgumentException
- void setChar(Object,char) throws IllegalAccessException, IllegalArgumentException
- void setDouble(Object,double) throws IllegalAccessException, IllegalArgumentException
- void setFloat(Object,float) throws IllegalAccessException, IllegalArgumentException
- void setInt(Object,int) throws IllegalAccessException, IllegalArgumentException
- void setLong(Object,long) throws IllegalAccessException, IllegalArgumentException
- void setShort(Object,short) throws IllegalAccessException, IllegalArgumentException
- String toString()

interface InvocationHandler

- abstract Object invoke(Object,Method, Object[]) throws Throwable

class InvocationTargetException extends Exception

- protected InvocationTargetException()
- InvocationTargetException(Throwable)
- InvocationTargetException(Throwable, String)
- Throwable getTargetException()
- void printStackTrace()
- void printStackTrace(java.io.PrintStream)
- void printStackTrace(java.io.PrintWriter)

interface Member

- final static int DECLARED
- abstract Class getDeclaringClass()
- abstract int getModifiers()
- abstract String getName()
- final static int PUBLIC

final class Method extends AccessibleObject implements Member

- boolean equals(Object)
- Class getDeclaringClass()
- Class[] getExceptionTypes()
- int getModifiers()
- String getName()
- Class[] getParameterTypes()
- Class getReturnType()
- int hashCode()
- Object invoke(Object,Object[]) throws IllegalAccessException, IllegalArgumentException, InvocationTargetException
- String toString()

class Modifier

- Modifier()
- final static int ABSTRACT

- final static int FINAL
- final static int INTERFACE
- static boolean isAbstract(int)
- static boolean isFinal(int)
- static boolean isInterface(int)
- static boolean isNative(int)
- static boolean isPrivate(int)
- static boolean isProtected(int)
- static boolean isPublic(int)
- static boolean isStatic(int)
- static boolean isStrict(int)
- static boolean isSynchronized(int)
- static boolean isTransient(int)
- static boolean isVolatile(int)
- final static int NATIVE
- final static int PRIVATE
- final static int PROTECTED
- final static int PUBLIC
- final static int STATIC
- final static int STRICT
- final static int SYNCHRONIZED
- static String toString(int)
- final static int TRANSIENT
- final static int VOLATILE

- **class Proxy implements java.io.Serializable**
- protected Proxy(InvocationHandler)
- static InvocationHandler getInvocationHandler(Object) throws IllegalArgumentException
- static Class getProxyClass(ClassLoader, Class[]) throws IllegalArgumentException
- protected InvocationHandler h
- static boolean isProxyClass(Class)
- static Object newProxyInstance(ClassLoader,Class[], InvocationHandler) throws IllegalArgumentException

- **final class ReflectPermission extends java.security.BasicPermission**
- ReflectPermission(String)
- ReflectPermission(String,String)

- **class UndeclaredThrowableException extends RuntimeException**
- UndeclaredThrowableException(Throwable)
- UndeclaredThrowableException(Throwable,String)
- Throwable getUndeclaredThrowable()
- void printStackTrace()
- void printStackTrace(java.io.PrintStream)
- void printStackTrace(java.io.PrintWriter)

22.3.5 java.math

- **package java.math**
- **class BigInteger extends Number implements Comparable**
- BigInteger(byte[])
- BigInteger(int,byte[])
- BigInteger(int,int,java.util.Random)
- BigInteger(int,java.util.Random)
- BigInteger(String)
- BigInteger(String,int)
- BigInteger abs()
- BigInteger add(BigInteger)
- BigInteger and(BigInteger)
- BigInteger andNot(BigInteger)
- int bitCount()
- int bitLength()
- BigInteger clearBit(int)
- int compareTo(Object)
- int compareTo(BigInteger)
- BigInteger divide(BigInteger)
- BigInteger[] divideAndRemainder(BigInteger)
- double doubleValue()
- boolean equals(Object)
- BigInteger flipBit(int)
- float floatValue()
- BigInteger gcd(BigInteger)
- int getLowestSetBit()
- int hashCode()
- int intValue()
- boolean isProbablePrime(int)
- long longValue()
- BigInteger max(BigInteger)
- BigInteger min(BigInteger)
- BigInteger mod(BigInteger)
- BigInteger modInverse(BigInteger)
- BigInteger modPow(BigInteger, BigInteger)
- BigInteger multiply(BigInteger)
- BigInteger negate()
- BigInteger not()
- final static BigInteger ONE
- BigInteger or(BigInteger)
- BigInteger pow(int)
- BigInteger remainder(BigInteger)
- BigInteger setBit(int)
- BigInteger shiftLeft(int)
- BigInteger shiftRight(int)
- int signum()
- BigInteger subtract(BigInteger)
- boolean testBit(int)
- byte[] toByteArray()
- String toString()
- String toString(int)
- static BigInteger valueOf(long)
- BigInteger xor(BigInteger)
- final static BigInteger ZERO

22.3.6 java.net

- **package java.net**
- **abstract class Authenticator**
- Authenticator()
- protected PasswordAuthentication getPasswordAuthentication()
- final protected int getRequestingPort()
- final protected String getRequestingPrompt()
- final protected String getRequestingProtocol()
- final protected String getRequestingScheme()
- final protected InetAddress getRequestingSite()

- static PasswordAuthentication requestPasswordAuthentication(InetAddress,int,String,String,String)
- static void setDefault(Authenticator)

class BindException extends SocketException

- BindException()
- BindException(String)

class ConnectException extends SocketException

- ConnectException()
- ConnectException(String)

abstract class ContentHandler

- ContentHandler()
- abstract Object getContent(URLConnection) throws java.io.IOException
- Object getContent(URLConnection, Class[]) throws java.io.IOException

interface ContentHandlerFactory

- abstract ContentHandler createContentHandler(String)

final class DatagramPacket

- DatagramPacket(byte[],int)
- DatagramPacket(byte[],int,int)
- DatagramPacket(byte[],int,int, InetAddress,int)
- DatagramPacket(byte[],int,InetAddress, int)
- InetAddress getAddress()
- byte[] getData()
- int getLength()
- int getOffset()
- int getPort()
- void setAddress(InetAddress)
- void setData(byte[])
- void setData(byte[],int,int)
- void setLength(int)
- void setPort(int)

class DatagramSocket

- DatagramSocket() throws SocketException
- DatagramSocket(int) throws SocketException
- DatagramSocket(int,InetAddress) throws SocketException
- void close()
- void connect(InetAddress,int)
- void disconnect()
- InetAddress getInetAddress()
- InetAddress getLocalAddress()
- int getLocalPort()
- int getPort()
- int getReceiveBufferSize() throws SocketException
- int getSendBufferSize() throws SocketException
- int getSoTimeout() throws SocketException
- void receive(DatagramPacket) throws java.io.IOException
- void send(DatagramPacket) throws java.io.IOException
- static void setDatagramSocketImplFactory(DatagramSocketImplFactory) throws java.io.IOException
- void setReceiveBufferSize(int) throws SocketException
- void setSendBufferSize(int) throws SocketException
- void setSoTimeout(int) throws SocketException

abstract class DatagramSocketImpl implements SocketOptions

- DatagramSocketImpl()
- abstract protected void bind(int, InetAddress) throws SocketException
- abstract protected void close()
- abstract protected void create() throws SocketException
- protected java.io.FileDescriptor fd
- protected java.io.FileDescriptor getFileDescriptor()
- protected int getLocalPort()
- abstract Object getOption(int) throws SocketException
- abstract protected int getTimeToLive() throws java.io.IOException
- abstract protected void join(InetAddress) throws java.io.IOException
- abstract protected void leave(InetAddress) throws java.io.IOException
- protected int localPort
- abstract protected int peek(InetAddress) throws java.io.IOException
- abstract protected void receive(DatagramPacket) throws java.io.IOException
- abstract protected void send(DatagramPacket) throws java.io.IOException
- abstract void setOption(int,Object) throws SocketException
- abstract protected void setTimeToLive(int) throws java.io.IOException

interface DatagramSocketImplFactory

- abstract DatagramSocketImpl createDatagramSocketImpl()

interface FileNameMap

- abstract String getContentTypeFor(String)

abstract class HttpURLConnection extends URLConnection

- protected HttpURLConnection(URL)
- abstract void disconnect()
- java.io.InputStream getErrorStream()
- static boolean getFollowRedirects()
- long getHeaderFieldDate(String,long)
- boolean getInstanceFollowRedirects()

- java.security.Permission getPermission() throws java.io.IOException
- String getRequestMethod()
- int getResponseCode() throws java.io.IOException
- String getResponseMessage() throws java.io.IOException
- final static int HTTP_ACCEPTED
- final static int HTTP_BAD_GATEWAY
- final static int HTTP_BAD_METHOD
- final static int HTTP_BAD_REQUEST
- final static int HTTP_CLIENT_TIMEOUT
- final static int HTTP_CONFLICT
- final static int HTTP_CREATED
- final static int HTTP_ENTITY_TOO_LARGE
- final static int HTTP_FORBIDDEN
- final static int HTTP_GATEWAY_TIMEOUT
- final static int HTTP_GONE
- final static int HTTP_INTERNAL_ERROR
- final static int HTTP_LENGTH_REQUIRED
- final static int HTTP_MOVED_PERM
- final static int HTTP_MOVED_TEMP
- final static int HTTP_MULT_CHOICE
- final static int HTTP_NO_CONTENT
- final static int HTTP_NOT_ACCEPTABLE
- final static int HTTP_NOT_AUTHORITATIVE
- final static int HTTP_NOT_FOUND
- final static int HTTP_NOT_IMPLEMENTED
- final static int HTTP_NOT_MODIFIED
- final static int HTTP_OK
- final static int HTTP_PARTIAL
- final static int HTTP_PAYMENT_REQUIRED
- final static int HTTP_PRECON_FAILED
- final static int HTTP_PROXY_AUTH
- final static int HTTP_REQ_TOO_LONG
- final static int HTTP_RESET
- final static int HTTP_SEE_OTHER
- final static int HTTP_UNAUTHORIZED
- final static int HTTP_UNAVAILABLE
- final static int HTTP_UNSUPPORTED_TYPE
- final static int HTTP_USE_PROXY
- final static int HTTP_VERSION
- protected boolean instanceFollowRedirects
- protected String method
- protected int responseCode
- protected String responseMessage
- static void setFollowRedirects(boolean)
- void setInstanceFollowRedirects(boolean)
- void setRequestMethod(String) throws ProtocolException
- abstract boolean usingProxy()

- **final class InetAddress implements java.io.Serializable**
- boolean equals(Object)
- byte[] getAddress()
- static InetAddress[] getAllByName(String) throws UnknownHostException
- static InetAddress getByName(String) throws UnknownHostException
- String getHostAddress()
- String getHostName()
- static InetAddress getLocalHost() throws UnknownHostException
- int hashCode()
- boolean isMulticastAddress()
- String toString()

- **abstract class JarURLConnection extends URLConnection**
- protected JarURLConnection(URL) throws MalformedURLException
- java.util.jar.Attributes getAttributes() throws java.io.IOException
- java.security.cert.Certificate[] getCertificates() throws java.io.IOException
- String getEntryName()
- java.util.jar.JarEntry getJarEntry() throws java.io.IOException
- abstract java.util.jar.JarFile getJarFile() throws java.io.IOException
- URL getJarFileURL()
- java.util.jar.Attributes getMainAttributes() throws java.io.IOException
- java.util.jar.Manifest getManifest() throws java.io.IOException
- protected URLConnection jarFileURLConnection

- **class MalformedURLException extends java.io.IOException**
- MalformedURLException()
- MalformedURLException(String)

- **class MulticastSocket extends DatagramSocket**
- MulticastSocket() throws java.io.IOException
- MulticastSocket(int) throws java.io.IOException
- InetAddress getInterface() throws SocketException
- int getTimeToLive() throws java.io.IOException
- void joinGroup(InetAddress) throws java.io.IOException
- void leaveGroup(InetAddress) throws java.io.IOException
- void send(DatagramPacket,byte) throws java.io.IOException
- void setInterface(InetAddress) throws SocketException
- void setTimeToLive(int) throws java.io.IOException

- **final class NetPermission extends java.security.BasicPermission**
- NetPermission(String)
- NetPermission(String,String)

- **class NoRouteToHostException extends SocketException**
- NoRouteToHostException()
- NoRouteToHostException(String)

- **final class PasswordAuthentication**
- PasswordAuthentication(String,char[])
- char[] getPassword()
- String getUserName()

- **class ProtocolException extends java.io.IOException**
- ProtocolException()
- ProtocolException(String)

- **class ServerSocket**
- ServerSocket(int) throws java.io.IOException

- ServerSocket(int,int) throws java.io.IOException
- ServerSocket(int,int,InetAddress) throws java.io.IOException
- Socket accept() throws java.io.IOException
- void close() throws java.io.IOException
- InetAddress getInetAddress()
- int getLocalPort()
- int getSoTimeout() throws java.io.IOException
- final protected void implAccept(Socket) throws java.io.IOException
- static void setSocketFactory(SocketImplFactory) throws java.io.IOException
- void setSoTimeout(int) throws SocketException
- String toString()

- **class Socket**
- protected Socket()
- Socket(String,int) throws UnknownHostException, java.io.IOException
- Socket(String,int,InetAddress,int) throws java.io.IOException
- Socket(InetAddress,int) throws java.io.IOException
- Socket(InetAddress,int,InetAddress,int) throws java.io.IOException
- protected Socket(SocketImpl) throws SocketException
- void close() throws java.io.IOException
- InetAddress getInetAddress()
- java.io.InputStream getInputStream() throws java.io.IOException
- boolean getKeepAlive() throws SocketException
- InetAddress getLocalAddress()
- int getLocalPort()
- java.io.OutputStream getOutputStream() throws java.io.IOException
- int getPort()
- int getReceiveBufferSize() throws SocketException
- int getSendBufferSize() throws SocketException
- int getSoLinger() throws SocketException
- int getSoTimeout() throws SocketException
- boolean getTcpNoDelay() throws SocketException
- void setKeepAlive(boolean) throws SocketException
- void setReceiveBufferSize(int) throws SocketException
- void setSendBufferSize(int) throws SocketException
- static void setSocketImplFactory(SocketImplFactory) throws java.io.IOException
- void setSoLinger(boolean,int) throws SocketException
- void setSoTimeout(int) throws SocketException
- void setTcpNoDelay(boolean) throws SocketException
- void shutdownInput() throws java.io.IOException
- void shutdownOutput() throws java.io.IOException
- String toString()

- **class SocketException extends java.io.IOException**
- SocketException()
- SocketException(String)

- **abstract class SocketImpl implements SocketOptions**
- SocketImpl()
- abstract protected void accept(SocketImpl) throws java.io.IOException
- protected InetAddress address
- abstract protected int available() throws java.io.IOException
- abstract protected void bind(InetAddress, int) throws java.io.IOException
- abstract protected void close() throws java.io.IOException
- abstract protected void connect(String, int) throws java.io.IOException
- abstract protected void connect(InetAddress,int) throws java.io.IOException
- abstract protected void create(boolean) throws java.io.IOException
- protected java.io.FileDescriptor fd
- protected java.io.FileDescriptor getFileDescriptor()
- protected InetAddress getInetAddress()
- abstract protected java.io.InputStream getInputStream() throws java.io.IOException
- protected int getLocalPort()
- abstract Object getOption(int) throws SocketException
- abstract protected java.io.OutputStream getOutputStream() throws java.io.IOException
- protected int getPort()
- abstract protected void listen(int) throws java.io.IOException
- protected int localport
- protected int port
- abstract void setOption(int,Object) throws SocketException
- protected void shutdownInput() throws java.io.IOException
- protected void shutdownOutput() throws java.io.IOException
- String toString()

- **interface SocketImplFactory**
- abstract SocketImpl createSocketImpl()

- **interface SocketOptions**
- abstract Object getOption(int) throws SocketException
- final static int IP_MULTICAST_IF
- abstract void setOption(int,Object) throws SocketException
- final static int SO_BINDADDR
- final static int SO_KEEPALIVE
- final static int SO_LINGER
- final static int SO_RCVBUF
- final static int SO_REUSEADDR

- final static int SO_SNDBUF
- final static int SO_TIMEOUT
- final static int TCP_NODELAY

- **final class SocketPermission extends java.security.Permission implements java.io.Serializable**
- SocketPermission(String,String)
- boolean equals(Object)
- String getActions()
- int hashCode()
- boolean implies(java.security.Permission)
- java.security.PermissionCollection newPermissionCollection()

- **class UnknownHostException extends java.io.IOException**
- UnknownHostException()
- UnknownHostException(String)

- **class UnknownServiceException extends java.io.IOException**
- UnknownServiceException()
- UnknownServiceException(String)

- **final class URL implements java.io.Serializable**
- URL(String) throws MalformedURLException
- URL(String,String,int,String) throws MalformedURLException
- URL(String,String,int,String, URLStreamHandler) throws MalformedURLException
- URL(String,String,String) throws MalformedURLException
- URL(URL,String) throws MalformedURLException
- URL(URL,String,URLStreamHandler) throws MalformedURLException
- boolean equals(Object)
- String getAuthority()
- final Object getContent() throws java.io.IOException
- final Object getContent(Class[]) throws java.io.IOException
- String getFile()
- String getHost()
- String getPath()
- int getPort()
- String getProtocol()
- String getQuery()
- String getRef()
- String getUserInfo()
- int hashCode()
- URLConnection openConnection() throws java.io.IOException
- final java.io.InputStream openStream() throws java.io.IOException
- boolean sameFile(URL)
- protected void set(String,String,int, String,String)
- protected void set(String,String,int, String,String,String,String,String)
- static void setURLStreamHandlerFactory(URLStream HandlerFactory)
- String toExternalForm()
- String toString()

- **class URLClassLoader extends java.security.SecureClassLoader**
- URLClassLoader(URL[])
- URLClassLoader(URL[],ClassLoader)
- URLClassLoader(URL[],ClassLoader, URLStreamHandlerFactory)
- protected void addURL(URL)
- protected Package definePackage(String, java.util.jar.Manifest,URL) throws IllegalArgumentException
- protected Class findClass(String) throws ClassNotFoundException
- URL findResource(String)
- java.util.Enumeration findResources(String) throws java.io.IOException
- protected java.security.PermissionCollection getPermissions(java.security.CodeSource)
- URL[] getURLs()
- static URLClassLoader newInstance(URL[])
- static URLClassLoader newInstance(URL[],ClassLoader)

- **abstract class URLConnection**
- protected URLConnection(URL)
- protected boolean allowUserInteraction
- abstract void connect() throws java.io.IOException
- protected boolean connected
- protected boolean doInput
- protected boolean doOutput
- boolean getAllowUserInteraction()
- Object getContent() throws java.io.IOException
- Object getContent(Class[]) throws java.io.IOException
- String getContentEncoding()
- int getContentLength()
- String getContentType()
- long getDate()
- static boolean getDefaultAllowUserInteraction()
- boolean getDefaultUseCaches()
- boolean getDoInput()
- boolean getDoOutput()
- long getExpiration()
- static FileNameMap getFileNameMap()
- String getHeaderField(int)
- String getHeaderField(String)
- long getHeaderFieldDate(String,long)
- int getHeaderFieldInt(String,int)
- String getHeaderFieldKey(int)
- long getIfModifiedSince()
- java.io.InputStream getInputStream() throws java.io.IOException
- long getLastModified()
- java.io.OutputStream getOutputStream() throws java.io.IOException
- java.security.Permission getPermission() throws java.io.IOException
- String getRequestProperty(String)
- URL getURL()
- boolean getUseCaches()
- protected static String guessContentTypeFromName(String)
- static String guessContentTypeFromStream(java.io.In putStream) throws java.io.IOException
- protected long ifModifiedSince
- void setAllowUserInteraction(boolean)

- static void setContentHandlerFactory(ContentHandlerFactory)
- static void setDefaultAllowUserInteraction(boolean)
- void setDefaultUseCaches(boolean)
- void setDoInput(boolean)
- void setDoOutput(boolean)
- static void setFileNameMap(FileNameMap)
- void setIfModifiedSince(long)
- void setRequestProperty(String,String)
- void setUseCaches(boolean)
- String toString()
- protected URL url
- protected boolean useCaches

- **class URLDecoder**
- URLDecoder()
- static String decode(String)

- **class URLEncoder**
- static String encode(String)

- **abstract class URLStreamHandler**
- URLStreamHandler()
- protected boolean equals(URL,URL)
- protected int getDefaultPort()
- protected InetAddress getHostAddress(URL)
- protected int hashCode(URL)
- protected boolean hostsEqual(URL,URL)
- abstract protected URLConnection openConnection(URL) throws java.io.IOException
- protected void parseURL(URL,String,int,int)
- protected boolean sameFile(URL,URL)
- protected void setURL(URL,String,String,int,String,String,String,String,String)
- protected String toExternalForm(URL)

- **interface URLStreamHandlerFactory**
- abstract URLStreamHandler createURLStreamHandler(String)

22.3.7 java.security

- **package java.security**
- **final class AccessControlContext**
- AccessControlContext(ProtectionDomain[])
- AccessControlContext(AccessControlContext,DomainCombiner)
- void checkPermission(Permission) throws AccessControlException
- boolean equals(Object)
- DomainCombiner getDomainCombiner()
- int hashCode()

- **class AccessControlException extends SecurityException**
- AccessControlException(String)
- AccessControlException(String, Permission)
- Permission getPermission()

- **final class AccessController**
- static void checkPermission(Permission) throws AccessControlException
- static Object doPrivileged(PrivilegedAction)
- static Object doPrivileged(PrivilegedAction, AccessControlContext)
- static Object doPrivileged(PrivilegedExceptionAction) throws PrivilegedActionException
- static Object doPrivileged(PrivilegedExceptionAction, AccessControlContext) throws PrivilegedActionException
- static AccessControlContext getContext()

- **class AlgorithmParameterGenerator**
- protected AlgorithmParameterGenerator(AlgorithmParameterGeneratorSpi,Provider,String)
- final AlgorithmParameters generateParameters()
- final String getAlgorithm()
- static AlgorithmParameterGenerator getInstance(String) throws NoSuchAlgorithmException
- static AlgorithmParameterGenerator getInstance(String,String) throws NoSuchAlgorithmException, NoSuchProviderException
- final Provider getProvider()
- final void init(int)
- final void init(int,SecureRandom)
- final void init(AlgorithmParameterSpec) throws InvalidAlgorithmParameterException
- final void init(AlgorithmParameterSpec, SecureRandom) throws InvalidAlgorithmParameterException

- **abstract class AlgorithmParameterGeneratorSpi**
- AlgorithmParameterGeneratorSpi()
- abstract protected AlgorithmParameters engineGenerateParameters()
- abstract protected void engineInit(int, SecureRandom)
- abstract protected void engineInit(AlgorithmParameterSpec, SecureRandom) throws InvalidAlgorithmParameterException

- **class AlgorithmParameters**
  - protected AlgorithmParameters(AlgorithmParametersSpi,Provider,String)
  - final String getAlgorithm()
  - final byte[] getEncoded() throws java.io.IOException
  - final byte[] getEncoded(String) throws java.io.IOException
  - static AlgorithmParameters getInstance(String) throws NoSuchAlgorithmException
  - static AlgorithmParameters getInstance(String,String) throws NoSuchAlgorithmException, NoSuchProviderException
  - final AlgorithmParameterSpec getParameterSpec(Class) throws InvalidParameterSpecException
  - final Provider getProvider()
  - final void init(byte[]) throws java.io.IOException
  - final void init(byte[],String) throws java.io.IOException
  - final void init(AlgorithmParameterSpec) throws InvalidParameterSpecException
  - final String toString()
- **abstract class AlgorithmParametersSpi**
  - AlgorithmParametersSpi()
  - abstract protected byte[] engineGetEncoded() throws java.io.IOException
  - abstract protected byte[] engineGetEncoded(String) throws java.io.IOException
  - abstract protected AlgorithmParameterSpec engineGetParameterSpec(Class) throws InvalidParameterSpecException
  - abstract protected void engineInit(byte[]) throws java.io.IOException
  - abstract protected void engineInit(byte[], String) throws java.io.IOException
  - abstract protected void engineInit(AlgorithmParameterSpec) throws InvalidParameterSpecException
  - abstract protected String engineToString()
- **final class AllPermission extends Permission**
  - AllPermission()
  - AllPermission(String,String)
  - boolean equals(Object)
  - String getActions()
  - int hashCode()
  - boolean implies(Permission)
  - PermissionCollection newPermissionCollection()
- **abstract class BasicPermission extends Permission implements java.io.Serializable**
  - BasicPermission(String)
  - BasicPermission(String,String)
  - boolean equals(Object)
  - String getActions()
  - int hashCode()
  - boolean implies(Permission)
  - PermissionCollection newPermissionCollection()
- **interface Certificate**
  - abstract void decode(java.io.InputStream) throws KeyException, java.io.IOException
  - abstract void encode(java.io.OutputStream) throws KeyException, java.io.IOException
  - abstract String getFormat()
  - abstract Principal getGuarantor()
  - abstract Principal getPrincipal()
  - abstract PublicKey getPublicKey()
  - abstract String toString(boolean)
- **class CodeSource implements java.io.Serializable**
  - CodeSource(java.net.URL,Certificate[])
  - boolean equals(Object)
  - final Certificate[] getCertificates()
  - final java.net.URL getLocation()
  - int hashCode()
  - boolean implies(CodeSource)
  - String toString()
- **class DigestException extends GeneralSecurityException**
  - DigestException()
  - DigestException(String)
- **class DigestInputStream extends java.io.FilterInputStream**
  - DigestInputStream(java.io.InputStream, MessageDigest)
  - protected MessageDigest digest
  - MessageDigest getMessageDigest()
  - void on(boolean)
  - int read() throws java.io.IOException
  - int read(byte[],int,int) throws java.io.IOException
  - void setMessageDigest(MessageDigest)
  - String toString()
- **class DigestOutputStream extends java.io.FilterOutputStream**
  - DigestOutputStream(java.io.OutputStream,MessageDigest)
  - protected MessageDigest digest
  - MessageDigest getMessageDigest()
  - void on(boolean)
  - void setMessageDigest(MessageDigest)
  - String toString()
  - void write(byte[],int,int) throws java.io.IOException
  - void write(int) throws java.io.IOException
- **interface DomainCombiner**
  - abstract ProtectionDomain[] combine(ProtectionDomain[], ProtectionDomain[])
- **class GeneralSecurityException extends Exception**
  - GeneralSecurityException()
  - GeneralSecurityException(String)

- **interface Guard**
  - abstract void checkGuard(Object) throws SecurityException
- **class GuardedObject implements java.io.Serializable**
  - GuardedObject(Object,Guard)
  - Object getObject() throws SecurityException
- **abstract class Identity implements Principal , java.io.Serializable**
  - protected Identity()
  - Identity(String)
  - Identity(String,IdentityScope) throws KeyManagementException
  - void addCertificate(Certificate) throws KeyManagementException
  - Certificate[] certificates()
  - final boolean equals(Object)
  - String getInfo()
  - final String getName()
  - PublicKey getPublicKey()
  - final IdentityScope getScope()
  - int hashCode()
  - protected boolean identityEquals(Identity)
  - void removeCertificate(Certificate) throws KeyManagementException
  - void setInfo(String)
  - void setPublicKey(PublicKey) throws KeyManagementException
  - String toString()
  - String toString(boolean)
- **abstract class IdentityScope extends Identity**
  - protected IdentityScope()
  - IdentityScope(String)
  - IdentityScope(String,IdentityScope) throws KeyManagementException
  - abstract void addIdentity(Identity) throws KeyManagementException
  - abstract Identity getIdentity(String)
  - Identity getIdentity(Principal)
  - abstract Identity getIdentity(PublicKey)
  - static IdentityScope getSystemScope()
  - abstract java.util.Enumeration identities()
  - abstract void removeIdentity(Identity) throws KeyManagementException
  - protected static void setSystemScope(IdentityScope)
  - abstract int size()
  - String toString()
- **class InvalidAlgorithmParameterException extends GeneralSecurityException**
  - InvalidAlgorithmParameterException()
  - InvalidAlgorithmParameterException(String)
- **class InvalidKeyException extends KeyException**
  - InvalidKeyException()
  - InvalidKeyException(String)
- **class InvalidParameterException extends IllegalArgumentException**
  - InvalidParameterException()
  - InvalidParameterException(String)
- **interface Key implements java.io.Serializable**
  - abstract String getAlgorithm()
  - abstract byte[] getEncoded()
  - abstract String getFormat()
  - final static long serialVersionUID
- **class KeyException extends GeneralSecurityException**
  - KeyException()
  - KeyException(String)
- **class KeyFactory**
  - protected KeyFactory(KeyFactorySpi, Provider,String)
  - final PrivateKey generatePrivate(KeySpec) throws InvalidKeySpecException
  - final PublicKey generatePublic(KeySpec) throws InvalidKeySpecException
  - final String getAlgorithm()
  - static KeyFactory getInstance(String) throws NoSuchAlgorithmException
  - static KeyFactory getInstance(String, String) throws NoSuchAlgorithmException, NoSuchProviderException
  - final KeySpec getKeySpec(Key,Class) throws InvalidKeySpecException
  - final Provider getProvider()
  - final Key translateKey(Key) throws InvalidKeyException
- **abstract class KeyFactorySpi**
  - KeyFactorySpi()
  - abstract protected PrivateKey engineGeneratePrivate(KeySpec) throws InvalidKeySpecException
  - abstract protected PublicKey engineGeneratePublic(KeySpec) throws InvalidKeySpecException
  - abstract protected KeySpec engineGetKeySpec(Key,Class) throws InvalidKeySpecException
  - abstract protected Key engineTranslateKey(Key) throws InvalidKeyException
- **class KeyManagementException extends KeyException**
  - KeyManagementException()
  - KeyManagementException(String)
- **final class KeyPair implements java.io.Serializable**
  - KeyPair(PublicKey,PrivateKey)
  - PrivateKey getPrivate()
  - PublicKey getPublic()
- **abstract class KeyPairGenerator extends KeyPairGeneratorSpi**
  - protected KeyPairGenerator(String)
  - KeyPair generateKeyPair()
  - final KeyPair genKeyPair()
  - String getAlgorithm()
  - static KeyPairGenerator getInstance(String) throws NoSuchAlgorithmException

- static KeyPairGenerator getInstance(String,String) throws NoSuchAlgorithmException, NoSuchProviderException
- final Provider getProvider()
- void initialize(int)
- void initialize(int,SecureRandom)
- void initialize(AlgorithmParameterSpec) throws InvalidAlgorithmParameterException
- void initialize(AlgorithmParameterSpec, SecureRandom) throws InvalidAlgorithmParameterException

- **abstract class KeyPairGeneratorSpi**
  - KeyPairGeneratorSpi()
  - abstract KeyPair generateKeyPair()
  - abstract void initialize(int, SecureRandom)
  - void initialize(AlgorithmParameterSpec, SecureRandom) throws InvalidAlgorithmParameterException

- **class KeyStore**
  - protected KeyStore(KeyStoreSpi, Provider,String)
  - final java.util.Enumeration aliases() throws KeyStoreException
  - final boolean containsAlias(String) throws KeyStoreException
  - final void deleteEntry(String) throws KeyStoreException
  - final Certificate getCertificate(String) throws KeyStoreException
  - final String getCertificateAlias(Certificate) throws KeyStoreException
  - final Certificate[] getCertificateChain(String) throws KeyStoreException
  - final java.util.Date getCreationDate(String) throws KeyStoreException
  - final static String getDefaultType()
  - static KeyStore getInstance(String) throws KeyStoreException
  - static KeyStore getInstance(String,String) throws KeyStoreException, NoSuchProviderException
  - final Key getKey(String,char[]) throws KeyStoreException, NoSuchAlgorithmException, UnrecoverableKeyException
  - final Provider getProvider()
  - final String getType()
  - final boolean isCertificateEntry(String) throws KeyStoreException
  - final boolean isKeyEntry(String) throws KeyStoreException
  - final void load(java.io.InputStream, char[]) throws java.io.IOException, NoSuchAlgorithmException, CertificateException
  - final void setCertificateEntry(String, Certificate) throws KeyStoreException
  - final void setKeyEntry(String,byte[], Certificate[]) throws KeyStoreException
  - final void setKeyEntry(String,Key,char[], Certificate[]) throws KeyStoreException
  - final int size() throws KeyStoreException
  - final void store(java.io.OutputStream, char[]) throws KeyStoreException, java.io.IOException, NoSuchAlgorithmException, CertificateException

- **class KeyStoreException extends GeneralSecurityException**
  - KeyStoreException()
  - KeyStoreException(String)

- **abstract class KeyStoreSpi**
  - KeyStoreSpi()
  - abstract java.util.Enumeration engineAliases()
  - abstract boolean engineContainsAlias(String)
  - abstract void engineDeleteEntry(String) throws KeyStoreException
  - abstract Certificate engineGetCertificate(String)
  - abstract String engineGetCertificateAlias(Certificate)
  - abstract Certificate[] engineGetCertificateChain(String)
  - abstract java.util.Date engineGetCreationDate(String)
  - abstract Key engineGetKey(String,char[]) throws NoSuchAlgorithmException, UnrecoverableKeyException
  - abstract boolean engineIsCertificateEntry(String)
  - abstract boolean engineIsKeyEntry(String)
  - abstract void engineLoad(java.io.InputStream,char[]) throws java.io.IOException, NoSuchAlgorithmException, CertificateException
  - abstract void engineSetCertificateEntry(String, Certificate) throws KeyStoreException
  - abstract void engineSetKeyEntry(String, byte[],Certificate[]) throws KeyStoreException
  - abstract void engineSetKeyEntry(String, Key,char[],Certificate[]) throws KeyStoreException
  - abstract int engineSize()
  - abstract void engineStore(java.io.OutputStream, char[]) throws java.io.IOException, NoSuchAlgorithmException, CertificateException

- **abstract class MessageDigest extends MessageDigestSpi**
  - protected MessageDigest(String)
  - Object clone() throws CloneNotSupportedException
  - byte[] digest()
  - byte[] digest(byte[])
  - int digest(byte[],int,int) throws DigestException
  - final String getAlgorithm()
  - final int getDigestLength()
  - static MessageDigest getInstance(String) throws NoSuchAlgorithmException

- static MessageDigest getInstance(String, String) throws NoSuchAlgorithmException, NoSuchProviderException
- final Provider getProvider()
- static boolean isEqual(byte[],byte[])
- void reset()
- String toString()
- void update(byte[])
- void update(byte[],int,int)
- void update(byte)

abstract class MessageDigestSpi

- MessageDigestSpi()
- Object clone() throws CloneNotSupportedException
- abstract protected byte[] engineDigest()
- protected int engineDigest(byte[],int,int) throws DigestException
- protected int engineGetDigestLength()
- abstract protected void engineReset()
- abstract protected void engineUpdate(byte[],int,int)
- abstract protected void engineUpdate(byte)

class NoSuchAlgorithmException extends GeneralSecurityException

- NoSuchAlgorithmException()
- NoSuchAlgorithmException(String)

class NoSuchProviderException extends GeneralSecurityException

- NoSuchProviderException()
- NoSuchProviderException(String)

abstract class Permission implements Guard , java.io.Serializable

- Permission(String)
- void checkGuard(Object) throws SecurityException
- abstract boolean equals(Object)
- abstract String getActions()
- final String getName()
- abstract int hashCode()
- abstract boolean implies(Permission)
- PermissionCollection newPermissionCollection()
- String toString()

abstract class PermissionCollection implements java.io.Serializable

- PermissionCollection()
- abstract void add(Permission)
- abstract java.util.Enumeration elements()
- abstract boolean implies(Permission)
- boolean isReadOnly()
- void setReadOnly()
- String toString()

final class Permissions extends PermissionCollection implements java.io.Serializable

- Permissions()
- void add(Permission)
- java.util.Enumeration elements()
- boolean implies(Permission)

abstract class Policy

- Policy()
- abstract PermissionCollection getPermissions(CodeSource)
- static Policy getPolicy()
- abstract void refresh()
- static void setPolicy(Policy)

interface Principal

- abstract boolean equals(Object)
- abstract String getName()
- abstract int hashCode()
- abstract String toString()

interface PrivateKey implements Key

- final static long serialVersionUID

interface PrivilegedAction

- abstract Object run()

class PrivilegedActionException extends Exception

- PrivilegedActionException(Exception)
- Exception getException()
- void printStackTrace()
- void printStackTrace(java.io.PrintStream)
- void printStackTrace(java.io.PrintWriter)
- String toString()

interface PrivilegedExceptionAction

- abstract Object run() throws Exception

class ProtectionDomain

- ProtectionDomain(CodeSource, PermissionCollection)
- final CodeSource getCodeSource()
- final PermissionCollection getPermissions()
- boolean implies(Permission)
- String toString()

abstract class Provider extends java.util.Properties

- protected Provider(String,double,String)
- void clear()
- java.util.Set entrySet()
- String getInfo()
- String getName()
- double getVersion()
- java.util.Set keySet()
- void load(java.io.InputStream) throws java.io.IOException
- Object put(Object,Object)
- void putAll(java.util.Map)
- Object remove(Object)
- String toString()
- java.util.Collection values()

class ProviderException extends RuntimeException

- ProviderException()
- ProviderException(String)

interface PublicKey implements Key

- final static long serialVersionUID

class SecureClassLoader extends ClassLoader

- protected SecureClassLoader()
- protected SecureClassLoader(ClassLoader)
- final protected Class defineClass(String, byte[],int,int,CodeSource)

- protected PermissionCollection getPermissions(CodeSource)

- **class SecureRandom extends java.util.Random**
  - SecureRandom()
  - SecureRandom(byte[])
  - protected SecureRandom(SecureRandomSpi, Provider)
  - byte[] generateSeed(int)
  - static SecureRandom getInstance(String) throws NoSuchAlgorithmException
  - static SecureRandom getInstance(String, String) throws NoSuchAlgorithmException, NoSuchProviderException
  - final Provider getProvider()
  - static byte[] getSeed(int)
  - final protected int next(int)
  - void nextBytes(byte[])
  - void setSeed(byte[])
  - void setSeed(long)

- **abstract class SecureRandomSpi implements java.io.Serializable**
  - SecureRandomSpi()
  - abstract protected byte[] engineGenerateSeed(int)
  - abstract protected void engineNextBytes(byte[])
  - abstract protected void engineSetSeed(byte[])

- **final class Security**
  - static int addProvider(Provider)
  - static String getProperty(String)
  - static Provider getProvider(String)
  - static Provider[] getProviders()
  - static Provider[] getProviders(String)
  - static Provider[] getProviders(java.util.Map)
  - static int insertProviderAt(Provider,int)
  - static void removeProvider(String)
  - static void setProperty(String,String)

- **final class SecurityPermission extends BasicPermission**
  - SecurityPermission(String)
  - SecurityPermission(String,String)

- **abstract class Signature extends SignatureSpi**
  - protected Signature(String)
  - Object clone() throws CloneNotSupportedException
  - final String getAlgorithm()
  - static Signature getInstance(String) throws NoSuchAlgorithmException
  - static Signature getInstance(String, String) throws NoSuchAlgorithmException, NoSuchProviderException
  - final Provider getProvider()
  - final void initSign(PrivateKey) throws InvalidKeyException
  - final void initSign(PrivateKey, SecureRandom) throws InvalidKeyException
  - final void initVerify(Certificate) throws InvalidKeyException
  - final void initVerify(PublicKey) throws InvalidKeyException
  - final void setParameter(AlgorithmParameterSpec) throws InvalidAlgorithmParameterException
  - final protected static int SIGN
  - final byte[] sign() throws SignatureException
  - final int sign(byte[],int,int) throws SignatureException
  - protected int state
  - String toString()
  - final protected static int UNINITIALIZED
  - final void update(byte[]) throws SignatureException
  - final void update(byte[],int,int) throws SignatureException
  - final void update(byte) throws SignatureException
  - final protected static int VERIFY
  - final boolean verify(byte[]) throws SignatureException

- **class SignatureException extends GeneralSecurityException**
  - SignatureException()
  - SignatureException(String)

- **abstract class SignatureSpi**
  - SignatureSpi()
  - protected SecureRandom appRandom
  - Object clone() throws CloneNotSupportedException
  - abstract protected void engineInitSign(PrivateKey) throws InvalidKeyException
  - protected void engineInitSign(PrivateKey, SecureRandom) throws InvalidKeyException
  - abstract protected void engineInitVerify(PublicKey) throws InvalidKeyException
  - protected void engineSetParameter(AlgorithmParameterSpec) throws InvalidAlgorithmParameterException
  - abstract protected byte[] engineSign() throws SignatureException
  - protected int engineSign(byte[],int,int) throws SignatureException
  - abstract protected void engineUpdate(byte[],int,int) throws SignatureException
  - abstract protected void engineUpdate(byte) throws SignatureException
  - abstract protected boolean engineVerify(byte[]) throws SignatureException

- **final class SignedObject implements java.io.Serializable**
  - SignedObject(java.io.Serializable, PrivateKey,Signature) throws java.io.IOException, InvalidKeyException, SignatureException

- String getAlgorithm()
- Object getObject() throws java.io.IOException, ClassNotFoundException
- byte[] getSignature()
- boolean verify(PublicKey,Signature) throws InvalidKeyException, SignatureException

abstract class Signer extends Identity

- protected Signer()
- Signer(String)
- Signer(String,IdentityScope) throws KeyManagementException
- PrivateKey getPrivateKey()
- final void setKeyPair(KeyPair) throws InvalidParameterException, KeyException
- String toString()

class UnrecoverableKeyException extends GeneralSecurityException

- UnrecoverableKeyException()
- UnrecoverableKeyException(String)

final class UnresolvedPermission extends Permission implements java.io.Serializable

- UnresolvedPermission(String,String, String,Certificate[])
- boolean equals(Object)
- String getActions()
- int hashCode()
- boolean implies(Permission)
- PermissionCollection newPermissionCollection()
- String toString()

22.3.8 java.security.acl

package java.security.acl

interface Acl implements Owner

- abstract boolean addEntry(java.security.Principal,AclEntry) throws NotOwnerException
- abstract boolean checkPermission(java.security.Principal, Permission)
- abstract java.util.Enumeration entries()
- abstract String getName()
- abstract java.util.Enumeration getPermissions(java.security.Principal)
- abstract boolean removeEntry(java.security.Principal, AclEntry) throws NotOwnerException
- abstract void setName(java.security.Principal,String) throws NotOwnerException
- abstract String toString()

interface AclEntry implements Cloneable

- abstract boolean addPermission(Permission)
- abstract boolean checkPermission(Permission)
- abstract Object clone()
- abstract java.security.Principal getPrincipal()
- abstract boolean isNegative()
- abstract java.util.Enumeration permissions()
- abstract boolean removePermission(Permission)
- abstract void setNegativePermissions()
- abstract boolean setPrincipal(java.security.Principal)
- abstract String toString()

class AclNotFoundException extends Exception

- AclNotFoundException()

interface Group implements java.security.Principal

- abstract boolean addMember(java.security.Principal)
- abstract boolean isMember(java.security.Principal)
- abstract java.util.Enumeration members()
- abstract boolean removeMember(java.security.Principal)

class LastOwnerException extends Exception

- LastOwnerException()

class NotOwnerException extends Exception

- NotOwnerException()

interface Owner

- abstract boolean addOwner(java.security.Principal, java.security.Principal) throws NotOwnerException
- abstract boolean deleteOwner(java.security.Principal, java.security.Principal) throws NotOwnerException, LastOwnerException
- abstract boolean isOwner(java.security.Principal)

interface Permission

- abstract boolean equals(Object)
- abstract String toString()

22.3.9 java.security.cert

package java.security.cert

abstract class Certificate implements java.io.Serializable

- protected Certificate(String)
- boolean equals(Object)
- abstract byte[] getEncoded() throws CertificateEncodingException
- abstract java.security.PublicKey getPublicKey()
- final String getType()
- int hashCode()
- abstract String toString()

- abstract void verify(java.security.PublicKey) throws CertificateException, java.security.NoSuchAlgorithmException, java.security.InvalidKeyException, java.security.NoSuchProviderException, java.security.SignatureException
- abstract void verify(java.security.PublicKey,String) throws CertificateException, java.security.NoSuchAlgorithmException, java.security.InvalidKeyException, java.security.NoSuchProviderException, java.security.SignatureException
- protected Object writeReplace() throws java.io.ObjectStreamException

- **class Certificate$CertificateRep implements java.io.Serializable**
- protected Certificate$CertificateRep(String,byte[])
- protected Object readResolve() throws java.io.ObjectStreamException

- **class CertificateEncodingException extends CertificateException**
- CertificateEncodingException()
- CertificateEncodingException(String)

- **class CertificateException extends java.security.GeneralSecurityException**
- CertificateException()
- CertificateException(String)

- **class CertificateExpiredException extends CertificateException**
- CertificateExpiredException()
- CertificateExpiredException(String)

- **class CertificateFactory**
- protected CertificateFactory(CertificateFactorySpi, java.security.Provider,String)
- final Certificate generateCertificate(java.io.InputStream) throws CertificateException
- final java.util.Collection generateCertificates(java.io.InputStream) throws CertificateException
- final CRL generateCRL(java.io.InputStream) throws CRLException
- final java.util.Collection generateCRLs(java.io.InputStream) throws CRLException
- final static CertificateFactory getInstance(String) throws CertificateException
- final static CertificateFactory getInstance(String,String) throws CertificateException, java.security.NoSuchProviderException
- final java.security.Provider getProvider()
- final String getType()

- **abstract class CertificateFactorySpi**
- CertificateFactorySpi()
- abstract Certificate engineGenerateCertificate(java.io.InputStream) throws CertificateException
- abstract java.util.Collection engineGenerateCertificates(java.io.InputStream) throws CertificateException
- abstract CRL engineGenerateCRL(java.io.InputStream) throws CRLException
- abstract java.util.Collection engineGenerateCRLs(java.io.InputStream) throws CRLException

- **class CertificateNotYetValidException extends CertificateException**
- CertificateNotYetValidException()
- CertificateNotYetValidException(String)

- **class CertificateParsingException extends CertificateException**
- CertificateParsingException()
- CertificateParsingException(String)

- **abstract class CRL**
- protected CRL(String)
- final String getType()
- abstract boolean isRevoked(Certificate)
- abstract String toString()

- **class CRLException extends java.security.GeneralSecurityException**
- CRLException()
- CRLException(String)

- **abstract class X509Certificate extends Certificate implements X509Extension**
- protected X509Certificate()
- abstract void checkValidity() throws CertificateExpiredException, CertificateNotYetValidException
- abstract void checkValidity(java.util.Date) throws CertificateExpiredException, CertificateNotYetValidException
- abstract int getBasicConstraints()
- abstract java.util.Set getCriticalExtensionOIDs()
- abstract byte[] getExtensionValue(String)
- abstract java.security.Principal getIssuerDN()
- abstract boolean[] getIssuerUniqueID()
- abstract boolean[] getKeyUsage()
- abstract java.util.Set getNonCriticalExtensionOIDs()
- abstract java.util.Date getNotAfter()
- abstract java.util.Date getNotBefore()
- abstract java.math.BigInteger getSerialNumber()
- abstract String getSigAlgName()
- abstract String getSigAlgOID()
- abstract byte[] getSigAlgParams()
- abstract byte[] getSignature()
- abstract java.security.Principal getSubjectDN()
- abstract boolean[] getSubjectUniqueID()
- abstract byte[] getTBSCertificate() throws CertificateEncodingException
- abstract int getVersion()
- abstract boolean hasUnsupportedCriticalExtension()

- **abstract class X509CRL extends CRL implements X509Extension**
- protected X509CRL()
- boolean equals(Object)
- abstract java.util.Set getCriticalExtensionOIDs()

- abstract byte[] getEncoded() throws CRLException
- abstract byte[] getExtensionValue(String)
- abstract java.security.Principal getIssuerDN()
- abstract java.util.Date getNextUpdate()
- abstract java.util.Set getNonCriticalExtensionOIDs()
- abstract X509CRLEntry getRevokedCertificate(java.math.BigInteger)
- abstract java.util.Set getRevokedCertificates()
- abstract String getSigAlgName()
- abstract String getSigAlgOID()
- abstract byte[] getSigAlgParams()
- abstract byte[] getSignature()
- abstract byte[] getTBSCertList() throws CRLException
- abstract java.util.Date getThisUpdate()
- abstract int getVersion()
- int hashCode()
- abstract boolean hasUnsupportedCriticalExtension()
- abstract void verify(java.security.PublicKey) throws CRLException, java.security.NoSuchAlgorithmException, java.security.InvalidKeyException, java.security.NoSuchProviderException, java.security.SignatureException
- abstract void verify(java.security.PublicKey,String) throws CRLException, java.security.NoSuchAlgorithmException, java.security.InvalidKeyException, java.security.NoSuchProviderException, java.security.SignatureException

abstract class X509CRLEntry implements X509Extension

- X509CRLEntry()
- boolean equals(Object)
- abstract java.util.Set getCriticalExtensionOIDs()
- abstract byte[] getEncoded() throws CRLException
- abstract byte[] getExtensionValue(String)
- abstract java.util.Set getNonCriticalExtensionOIDs()
- abstract java.util.Date getRevocationDate()
- abstract java.math.BigInteger getSerialNumber()
- abstract boolean hasExtensions()
- int hashCode()
- abstract boolean hasUnsupportedCriticalExtension()
- abstract String toString()

interface X509Extension

- abstract java.util.Set getCriticalExtensionOIDs()
- abstract byte[] getExtensionValue(String)
- abstract java.util.Set getNonCriticalExtensionOIDs()
- abstract boolean hasUnsupportedCriticalExtension()

22.3.10 java.security.interfaces

package java.security.interfaces

interface DSAKey

- abstract DSAParams getParams()

interface DSAKeyPairGenerator

- abstract void initialize(int,boolean, java.security.SecureRandom) throws java.security.InvalidParameterException
- abstract void initialize(DSAParams, java.security.SecureRandom) throws java.security.InvalidParameterException

interface DSAParams

- abstract java.math.BigInteger getG()
- abstract java.math.BigInteger getP()
- abstract java.math.BigInteger getQ()

interface DSAPrivateKey implements DSAKey , java.security.PrivateKey

- abstract java.math.BigInteger getX()
- final static long serialVersionUID

interface DSAPublicKey implements DSAKey , java.security.PublicKey

- abstract java.math.BigInteger getY()
- final static long serialVersionUID

interface RSAKey

- abstract java.math.BigInteger getModulus()

interface RSAPrivateCrtKey implements RSAPrivateKey

- abstract java.math.BigInteger getCrtCoefficient()
- abstract java.math.BigInteger getPrimeExponentP()
- abstract java.math.BigInteger getPrimeExponentQ()
- abstract java.math.BigInteger getPrimeP()
- abstract java.math.BigInteger getPrimeQ()
- abstract java.math.BigInteger getPublicExponent()

interface RSAPrivateKey implements java.security.PrivateKey , RSAKey

- abstract java.math.BigInteger getPrivateExponent()

interface RSAPublicKey implements java.security.PublicKey , RSAKey

- abstract java.math.BigInteger getPublicExponent()

22.3.11 java.security.spec

- **package java.security.spec**
- **interface AlgorithmParameterSpec**
- **class DSAParameterSpec implements AlgorithmParameterSpec , java.security.interfaces.DSAParams**
  - DSAParameterSpec(java.math.BigInteger, java.math.BigInteger, java.math.BigInteger)
  - java.math.BigInteger getG()
  - java.math.BigInteger getP()
  - java.math.BigInteger getQ()
- **class DSAPrivateKeySpec implements KeySpec**
  - DSAPrivateKeySpec(java.math.BigInteger, java.math.BigInteger, java.math.BigInteger, java.math.BigInteger)
  - java.math.BigInteger getG()
  - java.math.BigInteger getP()
  - java.math.BigInteger getQ()
  - java.math.BigInteger getX()
- **class DSAPublicKeySpec implements KeySpec**
  - DSAPublicKeySpec(java.math.BigInteger, java.math.BigInteger, java.math.BigInteger, java.math.BigInteger)
  - java.math.BigInteger getG()
  - java.math.BigInteger getP()
  - java.math.BigInteger getQ()
  - java.math.BigInteger getY()
- **abstract class EncodedKeySpec implements KeySpec**
  - EncodedKeySpec(byte[])
  - abstract String getFormat()
  - byte[] getEncoded()
- **class InvalidKeySpecException extends java.security.GeneralSecurityException**
  - InvalidKeySpecException()
  - InvalidKeySpecException(String)
- **class InvalidParameterSpecException extends java.security.GeneralSecurityException**
  - InvalidParameterSpecException()
  - InvalidParameterSpecException(String)
- **interface KeySpec**
- **class PKCS8EncodedKeySpec extends EncodedKeySpec**
  - PKCS8EncodedKeySpec(byte[])
  - final String getFormat()
  - byte[] getEncoded()
- **class RSAKeyGenParameterSpec implements AlgorithmParameterSpec**
  - RSAKeyGenParameterSpec(int, java.math.BigInteger)
  - final static java.math.BigInteger F4
  - int getKeysize()
  - final static java.math.BigInteger Fo
  - java.math.BigInteger getPublicExponent()
- **class RSAPrivateCrtKeySpec extends RSAPrivateKeySpec**
  - RSAPrivateCrtKeySpec(java.math.BigInteger,java.math.BigInteger, java.math.BigInteger, java.math.BigInteger, java.math.BigInteger, java.math.BigInteger, java.math.BigInteger, java.math.BigInteger)
  - java.math.BigInteger getCrtCoefficient()
  - java.math.BigInteger getPrimeExponentP()
  - java.math.BigInteger getPrimeExponentQ()
  - java.math.BigInteger getPrimeP()
  - java.math.BigInteger getPrimeQ()
  - java.math.BigInteger getPublicExponent()
- **class RSAPrivateKeySpec implements KeySpec**
  - RSAPrivateKeySpec(java.math.BigInteger, java.math.BigInteger)
  - java.math.BigInteger getModulus()
  - java.math.BigInteger getPrivateExponent()
- **class RSAPublicKeySpec implements KeySpec**
  - RSAPublicKeySpec(java.math.BigInteger, java.math.BigInteger)
  - java.math.BigInteger getModulus()
  - java.math.BigInteger getPublicExponent()
- **class X509EncodedKeySpec extends EncodedKeySpec**
  - X509EncodedKeySpec(byte[])
  - final String getFormat()
  - byte[] getEncoded()

22.3.12 java.text

- **package java.text**
- **class Annotation**
  - Annotation(Object)
  - String toString()
  - Object getValue()
- **interface AttributedCharacterIterator implements CharacterIterator**
  - abstract java.util.Set getAllAttributeKeys()
  - abstract int getRunLimit(AttributedCharacterIterator$Attribute)
  - abstract Object getAttribute(AttributedCharacterIterator$Attribute)
  - abstract int getRunLimit(java.util.Set)
  - abstract java.util.Map getAttributes()
  - abstract int getRunStart()
  - abstract int getRunLimit()
  - abstract int getRunStart(AttributedCharacterIterator$Attribute)
  - abstract int getRunStart(java.util.Set)

- **class AttributedCharacterIterator$Attribute implements java.io.Serializable**
  - protected AttributedCharacterIterator$Attribute(String)
  - final boolean equals(Object)
  - protected String getName()
  - final int hashCode()
  - final static AttributedCharacterIterator$Attribute INPUT_METHOD_SEGMENT
  - final static AttributedCharacterIterator$Attribute LANGUAGE
  - final static AttributedCharacterIterator$Attribute READING
  - protected Object readResolve() throws java.io.InvalidObjectException
  - String toString()
- **class AttributedString**
  - AttributedString(String)
  - AttributedString(String,java.util.Map)
  - AttributedString(AttributedCharacterIterator)
  - AttributedString(AttributedCharacterIterator,int,int)
  - AttributedString(AttributedCharacterIterator,int,int, AttributedCharacterIterator$Attribute[])
  - void addAttribute(AttributedCharacterIterator$Attribute,Object)
  - void addAttribute(AttributedCharacterIterator$Attribute,Object,int,int)
  - void addAttributes(java.util.Map,int,int)
  - AttributedCharacterIterator getIterator()
  - AttributedCharacterIterator getIterator(AttributedCharacterIterator$Attribute[])
  - AttributedCharacterIterator getIterator(AttributedCharacterIterator$Attribute[],int,int)
- **abstract class BreakIterator implements Cloneable**
  - protected BreakIterator()
  - Object clone()
  - abstract int current()
  - final static int DONE
  - abstract int first()
  - abstract int following(int)
  - static java.util.Locale[] getAvailableLocales()
  - static BreakIterator getCharacterInstance()
  - static BreakIterator getCharacterInstance(java.util.Locale)
  - static BreakIterator getLineInstance()
  - static BreakIterator getLineInstance(java.util.Locale)
  - static BreakIterator getSentenceInstance()
  - static BreakIterator getSentenceInstance(java.util.Locale)
  - abstract CharacterIterator getText()
  - static BreakIterator getWordInstance()
  - static BreakIterator getWordInstance(java.util.Locale)
  - boolean isBoundary(int)
  - abstract int last()
  - abstract int next()
  - abstract int next(int)
  - int preceding(int)
  - abstract int previous()
  - void setText(String)
  - abstract void setText(CharacterIterator)
- **interface CharacterIterator implements Cloneable**
  - abstract Object clone()
  - abstract char current()
  - final static char DONE
  - abstract char first()
  - abstract int getBeginIndex()
  - abstract int getEndIndex()
  - abstract int getIndex()
  - abstract char last()
  - abstract char next()
  - abstract char previous()
  - abstract char setIndex(int)
- **class ChoiceFormat extends NumberFormat**
  - ChoiceFormat(double[],String[])
  - ChoiceFormat(String)
  - void applyPattern(String)
  - Object clone()
  - boolean equals(Object)
  - StringBuffer format(double,StringBuffer, FieldPosition)
  - StringBuffer format(long,StringBuffer, FieldPosition)
  - Object[] getFormats()
  - double[] getLimits()
  - int hashCode()
  - final static double nextDouble(double)
  - static double nextDouble(double, boolean)
  - Number parse(String,ParsePosition)
  - final static double previousDouble(double)
  - void setChoices(double[],String[])
  - String toPattern()
- **final class CollationElementIterator**
  - int getMaxExpansion(int)
  - int getOffset()
  - int next()
  - final static int NULLORDER
  - int previous()
  - final static int primaryOrder(int)
  - void reset()
  - final static short secondaryOrder(int)
  - void setOffset(int)
  - void setText(String)
  - void setText(CharacterIterator)
  - final static short tertiaryOrder(int)
- **final class CollationKey implements Comparable**
  - int compareTo(Object)
  - int compareTo(CollationKey)
  - boolean equals(Object)
  - String getSourceString()
  - int hashCode()
  - byte[] toByteArray()

- **abstract class Collator implements java.util.Comparator , Cloneable**
  - protected Collator()
  - final static int CANONICAL_DECOMPOSITION
  - Object clone()
  - int compare(Object,Object)
  - abstract int compare(String,String)
  - boolean equals(Object)
  - boolean equals(String,String)
  - final static int FULL_DECOMPOSITION
  - static java.util.Locale[] getAvailableLocales()
  - abstract CollationKey getCollationKey(String)
  - int getDecomposition()
  - static Collator getInstance()
  - static Collator getInstance(java.util.Locale)
  - int getStrength()
  - abstract int hashCode()
  - final static int IDENTICAL
  - final static int NO_DECOMPOSITION
  - final static int PRIMARY
  - final static int SECONDARY
  - void setDecomposition(int)
  - void setStrength(int)
  - final static int TERTIARY
- **abstract class DateFormat extends Format**
  - protected DateFormat()
  - final static int AM_PM_FIELD
  - protected java.util.Calendar calendar
  - Object clone()
  - final static int DATE_FIELD
  - final static int DAY_OF_WEEK_FIELD
  - final static int DAY_OF_WEEK_IN_MONTH_FIELD
  - final static int DAY_OF_YEAR_FIELD
  - final static int DEFAULT
  - boolean equals(Object)
  - final static int ERA_FIELD
  - final StringBuffer format(Object, StringBuffer,FieldPosition)
  - final String format(java.util.Date)
  - abstract StringBuffer format(java.util.Date,StringBuffer, FieldPosition)
  - final static int FULL
  - static java.util.Locale[] getAvailableLocales()
  - java.util.Calendar getCalendar()
  - final static DateFormat getDateInstance()
  - final static DateFormat getDateInstance(int)
  - final static DateFormat getDateInstance(int,java.util.Locale)
  - final static DateFormat getDateTimeInstance()
  - final static DateFormat getDateTimeInstance(int,int)
  - final static DateFormat getDateTimeInstance(int,int, java.util.Locale)
  - final static DateFormat getInstance()
  - NumberFormat getNumberFormat()
  - final static DateFormat getTimeInstance()
  - final static DateFormat getTimeInstance(int)
  - final static DateFormat getTimeInstance(int,java.util.Locale)
  - java.util.TimeZone getTimeZone()
  - int hashCode()
  - final static int HOUR0_FIELD
  - final static int HOUR1_FIELD
  - final static int HOUR_OF_DAY0_FIELD
  - final static int HOUR_OF_DAY1_FIELD
  - boolean isLenient()
  - final static int LONG
  - final static int MEDIUM
  - final static int MILLISECOND_FIELD
  - final static int MINUTE_FIELD
  - final static int MONTH_FIELD
  - protected NumberFormat numberFormat
  - java.util.Date parse(String) throws ParseException
  - abstract java.util.Date parse(String, ParsePosition)
  - Object parseObject(String,ParsePosition)
  - final static int SECOND_FIELD
  - void setCalendar(java.util.Calendar)
  - void setLenient(boolean)
  - void setNumberFormat(NumberFormat)
  - void setTimeZone(java.util.TimeZone)
  - final static int SHORT
  - final static int TIMEZONE_FIELD
  - final static int WEEK_OF_MONTH_FIELD
  - final static int WEEK_OF_YEAR_FIELD
  - final static int YEAR_FIELD
- **class DateFormatSymbols implements java.io.Serializable , Cloneable**
  - DateFormatSymbols()
  - DateFormatSymbols(java.util.Locale)
  - Object clone()
  - boolean equals(Object)
  - String[] getAmPmStrings()
  - String[] getEras()
  - String getLocalPatternChars()
  - String[] getMonths()
  - String[] getShortMonths()
  - String[] getShortWeekdays()
  - String[] getWeekdays()
  - String[][] getZoneStrings()
  - int hashCode()
  - void setAmPmStrings(String[])
  - void setEras(String[])
  - void setLocalPatternChars(String)
  - void setMonths(String[])
  - void setShortMonths(String[])
  - void setShortWeekdays(String[])
  - void setWeekdays(String[])
  - void setZoneStrings(String[][])
- **class DecimalFormat extends NumberFormat**
  - DecimalFormat()
  - DecimalFormat(String)
  - DecimalFormat(String, DecimalFormatSymbols)
  - void applyLocalizedPattern(String)
  - void applyPattern(String)
  - Object clone()
  - boolean equals(Object)
  - StringBuffer format(double,StringBuffer, FieldPosition)
  - StringBuffer format(long,StringBuffer, FieldPosition)
  - DecimalFormatSymbols getDecimalFormatSymbols()
  - int getGroupingSize()
  - int getMultiplier()

- String getNegativePrefix()
- String getNegativeSuffix()
- String getPositivePrefix()
- String getPositiveSuffix()
- int hashCode()
- boolean isDecimalSeparatorAlwaysShown()
- Number parse(String,ParsePosition)
- void setDecimalFormatSymbols(DecimalFormatSymbols)
- void setDecimalSeparatorAlwaysShown(boolean)
- void setGroupingSize(int)
- void setMaximumFractionDigits(int)
- void setMaximumIntegerDigits(int)
- void setMinimumFractionDigits(int)
- void setMinimumIntegerDigits(int)
- void setMultiplier(int)
- void setNegativePrefix(String)
- void setNegativeSuffix(String)
- void setPositivePrefix(String)
- void setPositiveSuffix(String)
- String toLocalizedPattern()
- String toPattern()

- **final class DecimalFormatSymbols implements Cloneable , java.io.Serializable**
  - DecimalFormatSymbols()
  - DecimalFormatSymbols(java.util.Locale)
  - Object clone()
  - boolean equals(Object)
  - String getCurrencySymbol()
  - char getDecimalSeparator()
  - char getDigit()
  - char getGroupingSeparator()
  - String getInfinity()
  - String getInternationalCurrencySymbol()
  - char getMinusSign()
  - char getMonetaryDecimalSeparator()
  - String getNaN()
  - char getPatternSeparator()
  - char getPercent()
  - char getPerMill()
  - char getZeroDigit()
  - int hashCode()
  - void setCurrencySymbol(String)
  - void setDecimalSeparator(char)
  - void setDigit(char)
  - void setGroupingSeparator(char)
  - void setInfinity(String)
  - void setInternationalCurrencySymbol(String)
  - void setMinusSign(char)
  - void setMonetaryDecimalSeparator(char)
  - void setNaN(String)
  - void setPatternSeparator(char)
  - void setPercent(char)
  - void setPerMill(char)
  - void setZeroDigit(char)
- **class FieldPosition**
  - FieldPosition(int)
  - boolean equals(Object)
  - int getBeginIndex()
  - int getEndIndex()
  - int getField()
  - int hashCode()
  - void setBeginIndex(int)
  - void setEndIndex(int)
  - String toString()
- **abstract class Format implements java.io.Serializable , Cloneable**
  - Format()
  - Object clone()
  - final String format(Object)
  - abstract StringBuffer format(Object, StringBuffer,FieldPosition)
  - Object parseObject(String) throws ParseException
  - abstract Object parseObject(String, ParsePosition)
- **class MessageFormat extends Format**
  - MessageFormat(String)
  - void applyPattern(String)
  - Object clone()
  - boolean equals(Object)
  - final StringBuffer format(Object[], StringBuffer,FieldPosition)
  - final StringBuffer format(Object, StringBuffer,FieldPosition)
  - static String format(String,Object[])
  - Format[] getFormats()
  - java.util.Locale getLocale()
  - int hashCode()
  - Object[] parse(String) throws ParseException
  - Object[] parse(String,ParsePosition)
  - Object parseObject(String,ParsePosition)
  - void setFormat(int,Format)
  - void setFormats(Format[])
  - void setLocale(java.util.Locale)
  - String toPattern()
- **abstract class NumberFormat extends Format**
  - NumberFormat()
  - Object clone()
  - boolean equals(Object)
  - final String format(double)
  - abstract StringBuffer format(double, StringBuffer,FieldPosition)
  - final String format(long)
  - abstract StringBuffer format(long, StringBuffer,FieldPosition)
  - final StringBuffer format(Object, StringBuffer,FieldPosition)
  - final static int FRACTION_FIELD
  - static java.util.Locale[] getAvailableLocales()
  - final static NumberFormat getCurrencyInstance()
  - static NumberFormat getCurrencyInstance(java.util.Locale)
  - final static NumberFormat getInstance()
  - static NumberFormat getInstance(java.util.Locale)
  - int getMaximumFractionDigits()
  - int getMaximumIntegerDigits()
  - int getMinimumFractionDigits()
  - int getMinimumIntegerDigits()
  - final static NumberFormat getNumberInstance()
  - static NumberFormat getNumberInstance(java.util.Locale)
  - final static NumberFormat getPercentInstance()

- static NumberFormat getPercentInstance(java.util.Locale)
- int hashCode()
- final static int INTEGER_FIELD
- boolean isGroupingUsed()
- boolean isParseIntegerOnly()
- Number parse(String) throws ParseException
- abstract Number parse(String, ParsePosition)
- final Object parseObject(String, ParsePosition)
- void setGroupingUsed(boolean)
- void setMaximumFractionDigits(int)
- void setMaximumIntegerDigits(int)
- void setMinimumFractionDigits(int)
- void setMinimumIntegerDigits(int)
- void setParseIntegerOnly(boolean)

class ParseException extends Exception
- ParseException(String,int)
- int getErrorOffset()

class ParsePosition
- ParsePosition(int)
- boolean equals(Object)
- int getErrorIndex()
- int getIndex()
- int hashCode()
- void setErrorIndex(int)
- void setIndex(int)
- String toString()

class RuleBasedCollator extends Collator
- RuleBasedCollator(String) throws ParseException
- Object clone()
- int compare(String,String)
- boolean equals(Object)
- CollationElementIterator getCollationElementIterator(String)
- CollationElementIterator getCollationElementIterator(CharacterIterator)
- CollationKey getCollationKey(String)
- String getRules()
- int hashCode()

class SimpleDateFormat extends DateFormat
- SimpleDateFormat()
- SimpleDateFormat(String)
- SimpleDateFormat(String, DateFormatSymbols)
- SimpleDateFormat(String, java.util.Locale)
- void applyLocalizedPattern(String)
- void applyPattern(String)
- Object clone()
- boolean equals(Object)
- StringBuffer format(java.util.Date, StringBuffer,FieldPosition)
- java.util.Date get2DigitYearStart()
- DateFormatSymbols getDateFormatSymbols()
- int hashCode()
- java.util.Date parse(String,ParsePosition)
- void set2DigitYearStart(java.util.Date)
- void setDateFormatSymbols(DateFormatSymbols)
- String toLocalizedPattern()
- String toPattern()

final class StringCharacterIterator implements CharacterIterator
- StringCharacterIterator(String)
- StringCharacterIterator(String,int)
- StringCharacterIterator(String,int,int,int)
- Object clone()
- char current()
- boolean equals(Object)
- char first()
- int getBeginIndex()
- int getEndIndex()
- int getIndex()
- int hashCode()
- char last()
- char next()
- char previous()
- char setIndex(int)
- void setText(String)

22.3.13 java.text.resources

package java.text.resources

class BreakIteratorRules extends java.util.ListResourceBundle
- BreakIteratorRules()
- Object[][] getContents()

class BreakIteratorRules_th extends java.util.ListResourceBundle
- BreakIteratorRules_th()
- Object[][] getContents()

22.3.14 java.util

package java.util

abstract class AbstractCollection implements Collection
- protected AbstractCollection()
- boolean add(Object)
- boolean addAll(Collection)
- void clear()
- boolean contains(Object)
- boolean containsAll(Collection)
- boolean isEmpty()
- abstract Iterator iterator()
- boolean remove(Object)
- boolean removeAll(Collection)
- boolean retainAll(Collection)
- abstract int size()
- Object[] toArray()
- Object[] toArray(Object[])
- String toString()

abstract class AbstractList extends AbstractCollection implements List
- protected AbstractList()
- void add(int,Object)
- boolean add(Object)
- boolean addAll(int,Collection)

- void clear()
- boolean equals(Object)
- abstract Object get(int)
- int hashCode()
- int indexOf(Object)
- Iterator iterator()
- int lastIndexOf(Object)
- ListIterator listIterator()
- ListIterator listIterator(int)
- protected int modCount
- Object remove(int)
- protected void removeRange(int,int)
- Object set(int,Object)
- List subList(int,int)

abstract class AbstractMap implements Map

- protected AbstractMap()
- void clear()
- boolean containsKey(Object)
- boolean containsValue(Object)
- abstract Set entrySet()
- boolean equals(Object)
- Object get(Object)
- int hashCode()
- boolean isEmpty()
- Set keySet()
- Object put(Object,Object)
- void putAll(Map)
- Object remove(Object)
- int size()
- String toString()
- Collection values()

abstract class AbstractSequentialList extends AbstractList

- protected AbstractSequentialList()
- void add(int,Object)
- boolean addAll(int,Collection)
- Object get(int)
- Iterator iterator()
- abstract ListIterator listIterator(int)
- Object remove(int)
- Object set(int,Object)

abstract class AbstractSet extends AbstractCollection implements Set

- protected AbstractSet()
- boolean equals(Object)
- int hashCode()
- boolean removeAll(Collection)

class ArrayList extends AbstractList implements List , Cloneable , java.io.Serializable

- ArrayList()
- ArrayList(int)
- ArrayList(Collection)
- void add(int,Object)
- boolean add(Object)
- boolean addAll(int,Collection)
- boolean addAll(Collection)
- void clear()
- Object clone()
- boolean contains(Object)
- void ensureCapacity(int)
- Object get(int)
- int indexOf(Object)
- boolean isEmpty()
- int lastIndexOf(Object)
- Object remove(int)
- protected void removeRange(int,int)
- Object set(int,Object)
- int size()
- Object[] toArray()
- Object[] toArray(Object[])
- void trimToSize()

class Arrays

- static List asList(Object[])
- static int binarySearch(byte[],byte)
- static int binarySearch(char[],char)
- static int binarySearch(double[],double)
- static int binarySearch(float[],float)
- static int binarySearch(int[],int)
- static int binarySearch(long[],long)
- static int binarySearch(Object[],Object)
- static int binarySearch(Object[],Object, Comparator)
- static int binarySearch(short[],short)
- static boolean equals(byte[],byte[])
- static boolean equals(char[],char[])
- static boolean equals(double[],double[])
- static boolean equals(float[],float[])
- static boolean equals(int[],int[])
- static boolean equals(long[],long[])
- static boolean equals(Object[],Object[])
- static boolean equals(short[],short[])
- static boolean equals(boolean[], boolean[])
- static void fill(byte[],byte)
- static void fill(byte[],int,int,byte)
- static void fill(char[],char)
- static void fill(char[],int,int,char)
- static void fill(double[],double)
- static void fill(double[],int,int,double)
- static void fill(float[],float)
- static void fill(float[],int,int,float)
- static void fill(int[],int)
- static void fill(int[],int,int,int)
- static void fill(long[],int,int,long)
- static void fill(long[],long)
- static void fill(Object[],int,int,Object)
- static void fill(Object[],Object)
- static void fill(short[],int,int,short)
- static void fill(short[],short)
- static void fill(boolean[],int,int,boolean)
- static void fill(boolean[],boolean)
- static void sort(byte[])
- static void sort(byte[],int,int)
- static void sort(char[])
- static void sort(char[],int,int)
- static void sort(double[])
- static void sort(double[],int,int)
- static void sort(float[])
- static void sort(float[],int,int)
- static void sort(int[])
- static void sort(int[],int,int)
- static void sort(long[])
- static void sort(long[],int,int)
- static void sort(Object[])
- static void sort(Object[],int,int)
- static void sort(Object[],int,int, Comparator)
- static void sort(Object[],Comparator)
- static void sort(short[])
- static void sort(short[],int,int)

class BitSet implements java.io.Serializable , Cloneable

- BitSet()
- BitSet(int)
- void and(BitSet)
- void andNot(BitSet)
- void clear(int)
- Object clone()
- boolean equals(Object)
- boolean get(int)

- int hashCode()
- int length()
- void or(BitSet)
- void set(int)
- int size()
- String toString()
- void xor(BitSet)

abstract class Calendar implements java.io.Serializable , Cloneable

- protected Calendar()
- protected Calendar(TimeZone,Locale)
- abstract void add(int,int)
- boolean after(Object)
- final static int AM
- final static int AM_PM
- final static int APRIL
- protected boolean areFieldsSet
- final static int AUGUST
- boolean before(Object)
- final void clear()
- final void clear(int)
- Object clone()
- protected void complete()
- abstract protected void computeFields()
- abstract protected void computeTime()
- final static int DATE
- final static int DAY_OF_MONTH
- final static int DAY_OF_WEEK
- final static int DAY_OF_WEEK_IN_MONTH
- final static int DAY_OF_YEAR
- final static int DECEMBER
- final static int DST_OFFSET
- boolean equals(Object)
- final static int ERA
- final static int FEBRUARY
- final static int FIELD_COUNT
- protected int[] fields
- final static int FRIDAY
- final int get(int)
- int getActualMaximum(int)
- int getActualMinimum(int)
- static Locale[] getAvailableLocales()
- int getFirstDayOfWeek()
- abstract int getGreatestMinimum(int)
- static Calendar getInstance()
- static Calendar getInstance(Locale)
- static Calendar getInstance(TimeZone)
- static Calendar getInstance(TimeZone, Locale)
- abstract int getLeastMaximum(int)
- abstract int getMaximum(int)
- int getMinimalDaysInFirstWeek()
- abstract int getMinimum(int)
- final Date getTime()
- protected long getTimeInMillis()
- TimeZone getTimeZone()
- int hashCode()
- final static int HOUR
- final static int HOUR_OF_DAY
- final protected int internalGet(int)
- boolean isLenient()
- final boolean isSet(int)
- protected boolean[] isSet
- protected boolean isTimeSet
- final static int JANUARY
- final static int JULY
- final static int JUNE
- final static int MARCH
- final static int MAY
- final static int MILLISECOND
- final static int MINUTE
- final static int MONDAY
- final static int MONTH
- final static int NOVEMBER
- final static int OCTOBER
- final static int PM
- void roll(int,int)
- abstract void roll(int,boolean)
- final static int SATURDAY
- final static int SECOND
- final static int SEPTEMBER
- final void set(int,int)
- final void set(int,int,int)
- final void set(int,int,int,int,int)
- final void set(int,int,int,int,int,int)
- void setFirstDayOfWeek(int)
- void setLenient(boolean)
- void setMinimalDaysInFirstWeek(int)
- final void setTime(Date)
- protected void setTimeInMillis(long)
- void setTimeZone(TimeZone)
- final static int SUNDAY
- final static int THURSDAY
- protected long time
- String toString()
- final static int TUESDAY
- final static int UNDECIMBER
- final static int WEDNESDAY
- final static int WEEK_OF_MONTH
- final static int WEEK_OF_YEAR
- final static int YEAR
- final static int ZONE_OFFSET

interface Collection

- abstract boolean add(Object)
- abstract boolean addAll(Collection)
- abstract void clear()
- abstract boolean contains(Object)
- abstract boolean containsAll(Collection)
- abstract boolean equals(Object)
- abstract int hashCode()
- abstract boolean isEmpty()
- abstract Iterator iterator()
- abstract boolean remove(Object)
- abstract boolean removeAll(Collection)
- abstract boolean retainAll(Collection)
- abstract int size()
- abstract Object[] toArray()
- abstract Object[] toArray(Object[])

class Collections

- static int binarySearch(List,Object)
- static int binarySearch(List,Object, Comparator)
- static void copy(List,List)
- final static List EMPTY_LIST
- final static Map EMPTY_MAP
- final static Set EMPTY_SET
- static Enumeration enumeration(Collection)
- static void fill(List,Object)
- static Object max(Collection)
- static Object max(Collection, Comparator)
- static Object min(Collection)
- static Object min(Collection,Comparator)
- static List nCopies(int,Object)
- static void reverse(List)
- static Comparator reverseOrder()

- static void shuffle(List)
- static void shuffle(List,Random)
- static Set singleton(Object)
- static List singletonList(Object)
- static Map singletonMap(Object,Object)
- static void sort(List)
- static void sort(List,Comparator)
- static Collection synchronizedCollection(Collection)
- static List synchronizedList(List)
- static Map synchronizedMap(Map)
- static Set synchronizedSet(Set)
- static SortedMap synchronizedSortedMap(SortedMap)
- static SortedSet synchronizedSortedSet(SortedSet)
- static Collection unmodifiableCollection(Collection)
- static List unmodifiableList(List)
- static Map unmodifiableMap(Map)
- static Set unmodifiableSet(Set)
- static SortedMap unmodifiableSortedMap(SortedMap)
- static SortedSet unmodifiableSortedSet(SortedSet)

interface Comparator
- abstract int compare(Object,Object)
- abstract boolean equals(Object)

class ConcurrentModificationException extends RuntimeException
- ConcurrentModificationException()
- ConcurrentModificationException(String)

class Date implements java.io.Serializable , Cloneable , Comparable
- Date()
- Date(long)
- boolean after(Date)
- boolean before(Date)
- Object clone()
- int compareTo(Object)
- int compareTo(Date)
- boolean equals(Object)
- long getTime()
- int hashCode()
- void setTime(long)
- String toString()

abstract class Dictionary
- Dictionary()
- abstract Enumeration elements()
- abstract Object get(Object)
- abstract boolean isEmpty()
- abstract Enumeration keys()
- abstract Object put(Object,Object)
- abstract Object remove(Object)
- abstract int size()

class EmptyStackException extends RuntimeException
- EmptyStackException()

interface Enumeration
- abstract boolean hasMoreElements()
- abstract Object nextElement()

interface EventListener

class EventObject implements java.io.Serializable
- EventObject(Object)
- Object getSource()
- protected Object source
- String toString()

class GregorianCalendar extends Calendar
- GregorianCalendar()
- GregorianCalendar(int,int,int)
- GregorianCalendar(int,int,int,int,int)
- GregorianCalendar(int,int,int,int,int,int)
- GregorianCalendar(Locale)
- GregorianCalendar(TimeZone)
- GregorianCalendar(TimeZone,Locale)
- final static int AD
- void add(int,int)
- final static int BC
- protected void computeFields()
- protected void computeTime()
- boolean equals(Object)
- int getActualMaximum(int)
- int getActualMinimum(int)
- int getGreatestMinimum(int)
- final Date getGregorianChange()
- int getLeastMaximum(int)
- int getMaximum(int)
- int getMinimum(int)
- int hashCode()
- boolean isLeapYear(int)
- void roll(int,int)
- void roll(int,boolean)
- void setGregorianChange(Date)

class HashMap extends AbstractMap implements Map , Cloneable , java.io.Serializable
- HashMap()
- HashMap(int)
- HashMap(int,float)
- HashMap(Map)
- void clear()
- Object clone()
- boolean containsKey(Object)
- boolean containsValue(Object)
- Set entrySet()
- Object get(Object)
- boolean isEmpty()
- Set keySet()
- Object put(Object,Object)
- void putAll(Map)
- Object remove(Object)
- int size()
- Collection values()

class HashSet extends AbstractSet implements Set , Cloneable , java.io.Serializable
- HashSet()
- HashSet(int)
- HashSet(int,float)
- HashSet(Collection)
- boolean add(Object)
- void clear()
- Object clone()
- boolean contains(Object)
- boolean isEmpty()
- Iterator iterator()
- boolean remove(Object)
- int size()

class Hashtable extends Dictionary implements Map , Cloneable , java.io.Serializable
- Hashtable()
- Hashtable(int)

- Hashtable(int,float)
- Hashtable(Map)
- void clear()
- Object clone()
- boolean contains(Object)
- boolean containsKey(Object)
- boolean containsValue(Object)
- Enumeration elements()
- Set entrySet()
- boolean equals(Object)
- Object get(Object)
- int hashCode()
- boolean isEmpty()
- Enumeration keys()
- Set keySet()
- Object put(Object,Object)
- void putAll(Map)
- protected void rehash()
- Object remove(Object)
- int size()
- String toString()
- Collection values()

interface Iterator

- abstract boolean hasNext()
- abstract Object next()
- abstract void remove()

class LinkedList extends AbstractSequentialList implements List , Cloneable , java.io.Serializable

- LinkedList()
- LinkedList(Collection)
- void add(int,Object)
- boolean add(Object)
- boolean addAll(int,Collection)
- boolean addAll(Collection)
- void addFirst(Object)
- void addLast(Object)
- void clear()
- Object clone()
- boolean contains(Object)
- Object get(int)
- Object getFirst()
- Object getLast()
- int indexOf(Object)
- int lastIndexOf(Object)
- ListIterator listIterator(int)
- Object remove(int)
- boolean remove(Object)
- Object removeFirst()
- Object removeLast()
- Object set(int,Object)
- int size()
- Object[] toArray()
- Object[] toArray(Object[])

interface List implements Collection

- abstract void add(int,Object)
- abstract boolean add(Object)
- abstract boolean addAll(int,Collection)
- abstract boolean addAll(Collection)
- abstract void clear()
- abstract boolean contains(Object)
- abstract boolean containsAll(Collection)
- abstract boolean equals(Object)
- abstract Object get(int)
- abstract int hashCode()
- abstract int indexOf(Object)
- abstract boolean isEmpty()
- abstract Iterator iterator()
- abstract int lastIndexOf(Object)
- abstract ListIterator listIterator()
- abstract ListIterator listIterator(int)
- abstract Object remove(int)
- abstract boolean remove(Object)
- abstract boolean removeAll(Collection)
- abstract boolean retainAll(Collection)
- abstract Object set(int,Object)
- abstract int size()
- abstract List subList(int,int)
- abstract Object[] toArray()
- abstract Object[] toArray(Object[])

interface ListIterator implements Iterator

- abstract void add(Object)
- abstract boolean hasNext()
- abstract boolean hasPrevious()
- abstract Object next()
- abstract int nextIndex()
- abstract Object previous()
- abstract int previousIndex()
- abstract void remove()
- abstract void set(Object)

abstract class ListResourceBundle extends ResourceBundle

- ListResourceBundle()
- abstract protected Object[][] getContents()
- Enumeration getKeys()
- final Object handleGetObject(String)

final class Locale implements Cloneable , java.io.Serializable

- Locale(String,String)
- Locale(String,String,String)
- final static Locale CANADA
- final static Locale CANADA_FRENCH
- final static Locale CHINA
- final static Locale CHINESE
- Object clone()
- final static Locale ENGLISH
- boolean equals(Object)
- final static Locale FRANCE
- final static Locale FRENCH
- final static Locale GERMAN
- final static Locale GERMANY
- static Locale[] getAvailableLocales()
- String getCountry()
- static Locale getDefault()
- final String getDisplayCountry()
- String getDisplayCountry(Locale)
- final String getDisplayLanguage()
- String getDisplayLanguage(Locale)
- final String getDisplayName()
- String getDisplayName(Locale)
- final String getDisplayVariant()
- String getDisplayVariant(Locale)
- String getISO3Country() throws MissingResourceException
- String getISO3Language() throws MissingResourceException
- static String[] getISOCountries()
- static String[] getISOLanguages()
- String getLanguage()
- String getVariant()
- int hashCode()
- final static Locale ITALIAN
- final static Locale ITALY
- final static Locale JAPAN

- final static Locale JAPANESE
- final static Locale KOREA
- final static Locale KOREAN
- final static Locale PRC
- static void setDefault(Locale)
- final static Locale SIMPLIFIED_CHINESE
- final static Locale TAIWAN
- final String toString()
- final static Locale TRADITIONAL_CHINESE
- final static Locale UK
- final static Locale US

interface Map

- abstract void clear()
- abstract boolean containsKey(Object)
- abstract boolean containsValue(Object)
- abstract Set entrySet()
- abstract boolean equals(Object)
- abstract Object get(Object)
- abstract int hashCode()
- abstract boolean isEmpty()
- abstract Set keySet()
- abstract Object put(Object,Object)
- abstract void putAll(Map)
- abstract Object remove(Object)
- abstract int size()
- abstract Collection values()

interface Map$Entry

- abstract boolean equals(Object)
- abstract Object getKey()
- abstract Object getValue()
- abstract int hashCode()
- abstract Object setValue(Object)

class MissingResourceException extends RuntimeException

- MissingResourceException(String,String, String)
- String getClassName()
- String getKey()

class NoSuchElementException extends RuntimeException

- NoSuchElementException()
- NoSuchElementException(String)

class Observable

- Observable()
- void addObserver(Observer)
- protected void clearChanged()
- int countObservers()
- void deleteObserver(Observer)
- void deleteObservers()
- boolean hasChanged()
- void notifyObservers()
- void notifyObservers(Object)
- protected void setChanged()

interface Observer

- abstract void update(Observable,Object)

class Properties extends Hashtable

- Properties()
- Properties(Properties)
- protected Properties defaults
- String getProperty(String)
- String getProperty(String,String)
- void list(java.io.PrintStream)
- void list(java.io.PrintWriter)
- void load(java.io.InputStream) throws java.io.IOException
- Enumeration propertyNames()
- void save(java.io.OutputStream,String)
- Object setProperty(String,String)
- void store(java.io.OutputStream,String) throws java.io.IOException

final class PropertyPermission extends java.security.BasicPermission

- PropertyPermission(String,String)
- boolean equals(Object)
- String getActions()
- int hashCode()
- boolean implies(java.security.Permission)
- java.security.PermissionCollection newPermissionCollection()

class PropertyResourceBundle extends ResourceBundle

- PropertyResourceBundle(java.io.InputStream) throws java.io.IOException
- Enumeration getKeys()
- Object handleGetObject(String)

class Random implements java.io.Serializable

- Random()
- Random(long)
- protected int next(int)
- boolean nextBoolean()
- void nextBytes(byte[])
- double nextDouble()
- float nextFloat()
- double nextGaussian()
- int nextInt()
- int nextInt(int)
- long nextLong()
- void setSeed(long)

abstract class ResourceBundle

- ResourceBundle()
- final static ResourceBundle getBundle(String) throws MissingResourceException
- final static ResourceBundle getBundle(String,Locale)
- static ResourceBundle getBundle(String, Locale,ClassLoader) throws MissingResourceException
- abstract Enumeration getKeys()
- Locale getLocale()
- final Object getObject(String) throws MissingResourceException
- final String getString(String) throws MissingResourceException
- final String[] getStringArray(String) throws MissingResourceException
- abstract protected Object handleGetObject(String) throws MissingResourceException
- protected ResourceBundle parent
- protected void setParent(ResourceBundle)

interface Set implements Collection

- abstract boolean add(Object)
- abstract boolean addAll(Collection)

- abstract void clear()
- abstract boolean contains(Object)
- abstract boolean containsAll(Collection)
- abstract boolean equals(Object)
- abstract int hashCode()
- abstract boolean isEmpty()
- abstract Iterator iterator()
- abstract boolean remove(Object)
- abstract boolean removeAll(Collection)
- abstract boolean retainAll(Collection)
- abstract int size()
- abstract Object[] toArray()
- abstract Object[] toArray(Object[])

class SimpleTimeZone extends TimeZone

- SimpleTimeZone(int,String)
- SimpleTimeZone(int,String,int,int,int,int, int,int,int,int)
- SimpleTimeZone(int,String,int,int,int,int, int,int,int,int,int)
- Object clone()
- boolean equals(Object)
- int getDSTSavings()
- int getOffset(int,int,int,int,int,int)
- int getRawOffset()
- int hashCode()
- boolean hasSameRules(TimeZone)
- boolean inDaylightTime(Date)
- void setDSTSavings(int)
- void setEndRule(int,int,int)
- void setEndRule(int,int,int,int)
- void setEndRule(int,int,int,int,boolean)
- void setRawOffset(int)
- void setStartRule(int,int,int)
- void setStartRule(int,int,int,int)
- void setStartRule(int,int,int,int,boolean)
- void setStartYear(int)
- String toString()
- boolean useDaylightTime()

interface SortedMap implements Map

- abstract Comparator comparator()
- abstract Object firstKey()
- abstract SortedMap headMap(Object)
- abstract Object lastKey()
- abstract SortedMap subMap(Object, Object)
- abstract SortedMap tailMap(Object)

interface SortedSet implements Set

- abstract Comparator comparator()
- abstract Object first()
- abstract SortedSet headSet(Object)
- abstract Object last()
- abstract SortedSet subSet(Object, Object)
- abstract SortedSet tailSet(Object)

class Stack extends Vector

- Stack()
- boolean empty()
- Object peek()
- Object pop()
- Object push(Object)
- int search(Object)

class StringTokenizer implements Enumeration

- StringTokenizer(String)
- StringTokenizer(String,String)
- StringTokenizer(String,String,boolean)
- int countTokens()
- boolean hasMoreElements()
- boolean hasMoreTokens()
- Object nextElement()
- String nextToken()
- String nextToken(String)

class Timer

- Timer()
- Timer(boolean)
- void cancel()
- void schedule(TimerTask,long)
- void schedule(TimerTask,long,long)
- void schedule(TimerTask,Date)
- void schedule(TimerTask,Date,long)
- void scheduleAtFixedRate(TimerTask, long,long)
- void scheduleAtFixedRate(TimerTask, Date,long)

abstract class TimerTask implements Runnable

- protected TimerTask()
- boolean cancel()
- abstract void run()
- long scheduledExecutionTime()

abstract class TimeZone implements java.io.Serializable , Cloneable

- TimeZone()
- Object clone()
- static String[] getAvailableIDs()
- static String[] getAvailableIDs(int)
- static TimeZone getDefault()
- final String getDisplayName()
- final String getDisplayName(Locale)
- final String getDisplayName(boolean,int)
- String getDisplayName(boolean,int, Locale)
- String getID()
- abstract int getOffset(int,int,int,int,int, int)
- abstract int getRawOffset()
- static TimeZone getTimeZone(String)
- boolean hasSameRules(TimeZone)
- abstract boolean inDaylightTime(Date)
- final static int LONG
- static void setDefault(TimeZone)
- void setID(String)
- abstract void setRawOffset(int)
- final static int SHORT
- abstract boolean useDaylightTime()

class TooManyListenersException extends Exception

- TooManyListenersException()
- TooManyListenersException(String)

class TreeMap extends AbstractMap implements SortedMap , Cloneable , java.io.Serializable

- TreeMap()
- TreeMap(Comparator)
- TreeMap(Map)
- TreeMap(SortedMap)
- void clear()
- Object clone()
- Comparator comparator()
- boolean containsKey(Object)

- boolean containsValue(Object)
- Set entrySet()
- Object firstKey()
- Object get(Object)
- SortedMap headMap(Object)
- Set keySet()
- Object lastKey()
- Object put(Object,Object)
- void putAll(Map)
- Object remove(Object)
- int size()
- SortedMap subMap(Object,Object)
- SortedMap tailMap(Object)
- Collection values()

class TreeSet extends AbstractSet implements SortedSet , Cloneable , java.io.Serializable

- TreeSet()
- TreeSet(Collection)
- TreeSet(Comparator)
- TreeSet(SortedSet)
- boolean add(Object)
- boolean addAll(Collection)
- void clear()
- Object clone()
- Comparator comparator()
- boolean contains(Object)
- Object first()
- SortedSet headSet(Object)
- boolean isEmpty()
- Iterator iterator()
- Object last()
- boolean remove(Object)
- int size()
- SortedSet subSet(Object,Object)
- SortedSet tailSet(Object)

class Vector extends AbstractList implements List , Cloneable , java.io.Serializable

- Vector()
- Vector(int)
- Vector(int,int)
- Vector(Collection)
- void add(int,Object)
- boolean add(Object)
- boolean addAll(int,Collection)
- boolean addAll(Collection)
- void addElement(Object)
- int capacity()
- protected int capacityIncrement
- void clear()
- Object clone()
- boolean contains(Object)
- boolean containsAll(Collection)
- void copyInto(Object[])
- Object elementAt(int)
- protected int elementCount
- protected Object[] elementData
- Enumeration elements()
- void ensureCapacity(int)
- boolean equals(Object)
- Object firstElement()
- Object get(int)
- int hashCode()
- int indexOf(Object)
- int indexOf(Object,int)
- void insertElementAt(Object,int)
- boolean isEmpty()
- Object lastElement()
- int lastIndexOf(Object)
- int lastIndexOf(Object,int)
- Object remove(int)
- boolean remove(Object)
- boolean removeAll(Collection)
- void removeAllElements()
- boolean removeElement(Object)
- void removeElementAt(int)
- protected void removeRange(int,int)
- boolean retainAll(Collection)
- Object set(int,Object)
- void setElementAt(Object,int)
- void setSize(int)
- int size()
- List subList(int,int)
- Object[] toArray()
- Object[] toArray(Object[])
- String toString()
- void trimToSize()

class WeakHashMap extends AbstractMap implements Map

- WeakHashMap()
- WeakHashMap(int)
- WeakHashMap(int,float)
- WeakHashMap(Map)
- void clear()
- boolean containsKey(Object)
- Set entrySet()
- Object get(Object)
- boolean isEmpty()
- Object put(Object,Object)
- Object remove(Object)
- int size()

22.3.15 java.util.jar

package java.util.jar

class Attributes implements Cloneable , java.util.Map

- Attributes()
- Attributes(int)
- Attributes(Attributes)
- void clear()
- Object clone()
- boolean containsKey(Object)
- boolean containsValue(Object)
- java.util.Set entrySet()
- boolean equals(Object)
- Object get(Object)
- String getValue(String)
- String getValue(Attributes$Name)
- int hashCode()
- boolean isEmpty()
- java.util.Set keySet()
- protected java.util.Map map
- Object put(Object,Object)
- void putAll(java.util.Map)
- String putValue(String,String)
- Object remove(Object)
- int size()
- java.util.Collection values()

class Attributes$Name

- Attributes$Name(String)
- final static Attributes$Name CLASS_PATH
- final static Attributes$Name CONTENT_TYPE
- boolean equals(Object)

- final static Attributes$Name EXTENSION_INSTALLATION
- final static Attributes$Name EXTENSION_LIST
- final static Attributes$Name EXTENSION_NAME
- int hashCode()
- final static Attributes$Name IMPLEMENTATION_TITLE
- final static Attributes$Name IMPLEMENTATION_URL
- final static Attributes$Name IMPLEMENTATION_VENDOR
- final static Attributes$Name IMPLEMENTATION_VENDOR_ID
- final static Attributes$Name IMPLEMENTATION_VERSION
- final static Attributes$Name MAIN_CLASS
- final static Attributes$Name MANIFEST_VERSION
- final static Attributes$Name SEALED
- final static Attributes$Name SIGNATURE_VERSION
- final static Attributes$Name SPECIFICATION_TITLE
- final static Attributes$Name SPECIFICATION_VENDOR
- final static Attributes$Name SPECIFICATION_VERSION
- String toString()

- **class JarEntry extends java.util.zip.ZipEntry**
- JarEntry(String)
- JarEntry(JarEntry)
- JarEntry(java.util.zip.ZipEntry)
- Attributes getAttributes() throws java.io.IOException
- java.security.cert.Certificate[] getCertificates()

- **class JarException extends java.util.zip.ZipException**
- JarException()
- JarException(String)

- **class JarFile extends java.util.zip.ZipFile**
- JarFile(java.io.File) throws java.io.IOException
- JarFile(java.io.File,boolean) throws java.io.IOException
- JarFile(java.io.File,boolean,int) throws java.io.IOException
- JarFile(String) throws java.io.IOException
- JarFile(String,boolean) throws java.io.IOException
- java.util.Enumeration entries()
- java.util.zip.ZipEntry getEntry(String)
- java.io.InputStream getInputStream(java.util.zip.ZipEntry) throws java.io.IOException
- JarEntry getJarEntry(String)
- Manifest getManifest() throws java.io.IOException
- final static String MANIFEST_NAME

- **class JarInputStream extends java.util.zip.ZipInputStream**
- JarInputStream(java.io.InputStream) throws java.io.IOException
- JarInputStream(java.io.InputStream, boolean) throws java.io.IOException
- protected java.util.zip.ZipEntry createZipEntry(String)
- Manifest getManifest()
- java.util.zip.ZipEntry getNextEntry() throws java.io.IOException
- JarEntry getNextJarEntry() throws java.io.IOException
- int read(byte[],int,int) throws java.io.IOException

- **class JarOutputStream extends java.util.zip.ZipOutputStream**
- JarOutputStream(java.io.OutputStream) throws java.io.IOException
- JarOutputStream(java.io.OutputStream, Manifest) throws java.io.IOException
- void putNextEntry(java.util.zip.ZipEntry) throws java.io.IOException

- **class Manifest implements Cloneable**
- Manifest()
- Manifest(java.io.InputStream) throws java.io.IOException
- Manifest(Manifest)
- void clear()
- Object clone()
- boolean equals(Object)
- Attributes getAttributes(String)
- java.util.Map getEntries()
- Attributes getMainAttributes()
- int hashCode()
- void read(java.io.InputStream) throws java.io.IOException
- void write(java.io.OutputStream) throws java.io.IOException

22.3.16 java.util.zip

- **package java.util.zip**
- **class Adler32 implements Checksum**
- Adler32()
- long getValue()
- void reset()
- void update(byte[])
- void update(byte[],int,int)
- void update(int)

- **class CheckedInputStream extends java.io.FilterInputStream**
- CheckedInputStream(java.io.InputStream ,Checksum)
- Checksum getChecksum()
- int read() throws java.io.IOException
- int read(byte[],int,int) throws java.io.IOException
- long skip(long) throws java.io.IOException

- **class CheckedOutputStream extends java.io.FilterOutputStream**

- CheckedOutputStream(java.io.OutputStream,Checksum)
- Checksum getChecksum()

interface Checksum
- abstract long getValue()
- abstract void reset()
- abstract void update(byte[],int,int)
- abstract void update(int)

class CRC32 implements Checksum
- CRC32()
- long getValue()
- void reset()
- void update(byte[])
- void update(byte[],int,int)
- void update(int)

class DataFormatException extends Exception
- DataFormatException()
- DataFormatException(String)

class Deflater
- Deflater()
- Deflater(int)
- Deflater(int,boolean)
- final static int BEST_COMPRESSION
- final static int BEST_SPEED
- final static int DEFAULT_COMPRESSION
- final static int DEFAULT_STRATEGY
- int deflate(byte[])
- int deflate(byte[],int,int)
- final static int DEFLATED
- void end()
- final static int FILTERED
- protected void finalize()
- void finish()
- boolean finished()
- int getAdler()
- int getTotalIn()
- int getTotalOut()
- final static int HUFFMAN_ONLY
- boolean needsInput()
- final static int NO_COMPRESSION
- void reset()
- void setDictionary(byte[])
- void setDictionary(byte[],int,int)
- void setInput(byte[])
- void setInput(byte[],int,int)
- void setLevel(int)
- void setStrategy(int)

class DeflaterOutputStream extends java.io.FilterOutputStream
- DeflaterOutputStream(java.io.OutputStream)
- DeflaterOutputStream(java.io.OutputStream,Deflater)
- DeflaterOutputStream(java.io.OutputStream,Deflater,int)
- protected byte[] buf
- void close() throws java.io.IOException
- protected Deflater def
- protected void deflate() throws java.io.IOException
- void finish() throws java.io.IOException
- void write(byte[],int,int) throws java.io.IOException
- void write(int) throws java.io.IOException

class GZIPInputStream extends InflaterInputStream
- GZIPInputStream(java.io.InputStream) throws java.io.IOException
- GZIPInputStream(java.io.InputStream,int) throws java.io.IOException
- void close() throws java.io.IOException
- protected CRC32 crc
- protected boolean eos
- final static int GZIP_MAGIC
- int read(byte[],int,int) throws java.io.IOException

class GZIPOutputStream extends DeflaterOutputStream
- GZIPOutputStream(java.io.OutputStream) throws java.io.IOException
- GZIPOutputStream(java.io.OutputStream,int) throws java.io.IOException
- void close() throws java.io.IOException
- protected CRC32 crc
- void finish() throws java.io.IOException
- void write(byte[],int,int) throws java.io.IOException

class Inflater
- Inflater()
- Inflater(boolean)
- void end()
- protected void finalize()
- boolean finished()
- int getAdler()
- int getRemaining()
- int getTotalIn()
- int getTotalOut()
- int inflate(byte[]) throws DataFormatException
- int inflate(byte[],int,int) throws DataFormatException
- boolean needsDictionary()
- boolean needsInput()
- void reset()
- void setDictionary(byte[])
- void setDictionary(byte[],int,int)
- void setInput(byte[])
- void setInput(byte[],int,int)

class InflaterInputStream extends java.io.FilterInputStream
- InflaterInputStream(java.io.InputStream)
- InflaterInputStream(java.io.InputStream, Inflater)
- InflaterInputStream(java.io.InputStream, Inflater,int)
- int available() throws java.io.IOException
- protected byte[] buf
- void close() throws java.io.IOException
- protected void fill() throws java.io.IOException
- protected Inflater inf
- protected int len

- int read() throws java.io.IOException
- int read(byte[],int,int) throws java.io.IOException
- long skip(long) throws java.io.IOException

class ZipEntry implements ZipConstants , Cloneable

- ZipEntry(String)
- ZipEntry(ZipEntry)
- Object clone()
- final static int DEFLATED
- String getComment()
- long getCompressedSize()
- long getCrc()
- byte[] getExtra()
- int getMethod()
- String getName()
- long getSize()
- long getTime()
- int hashCode()
- boolean isDirectory()
- void setComment(String)
- void setCompressedSize(long)
- void setCrc(long)
- void setExtra(byte[])
- void setMethod(int)
- void setSize(long)
- void setTime(long)
- final static int STORED
- String toString()

class ZipException extends java.io.IOException

- ZipException()
- ZipException(String)

class ZipFile implements ZipConstants

- ZipFile(java.io.File) throws ZipException, java.io.IOException
- ZipFile(java.io.File,int) throws java.io.IOException
- ZipFile(String) throws java.io.IOException
- void close() throws java.io.IOException
- java.util.Enumeration entries()
- protected void finalize() throws java.io.IOException
- ZipEntry getEntry(String)
- java.io.InputStream getInputStream(ZipEntry) throws java.io.IOException
- String getName()
- final static int OPEN_DELETE
- final static int OPEN_READ
- int size()

class ZipInputStream extends InflaterInputStream implements ZipConstants

- ZipInputStream(java.io.InputStream)
- int available() throws java.io.IOException
- void close() throws java.io.IOException
- void closeEntry() throws java.io.IOException
- protected ZipEntry createZipEntry(String)
- ZipEntry getNextEntry() throws java.io.IOException
- int read(byte[],int,int) throws java.io.IOException
- long skip(long) throws java.io.IOException

class ZipOutputStream extends DeflaterOutputStream implements ZipConstants

- ZipOutputStream(java.io.OutputStream)
- void close() throws java.io.IOException
- void closeEntry() throws java.io.IOException
- final static int DEFLATED
- void finish() throws java.io.IOException
- void putNextEntry(ZipEntry) throws java.io.IOException
- void setComment(String)
- void setLevel(int)
- void setMethod(int)
- final static int STORED
- void write(byte[],int,int) throws java.io.IOException

22.3.17 javax.microedition.io

package javax.microedition.io

interface Connection

- abstract void close() throws java.io.IOException

class ConnectionNotFoundException extends java.io.IOException

- ConnectionNotFoundException()
- ConnectionNotFoundException(String)

class Connector

- static Connection open(String) throws java.io.IOException
- static Connection open(String,int) throws java.io.IOException
- static Connection open(String,int, boolean) throws java.io.IOException
- static java.io.DataInputStream openDataInputStream(String) throws java.io.IOException
- static java.io.DataOutputStream openDataOutputStream(String) throws java.io.IOException
- static java.io.InputStream openInputStream(String) throws java.io.IOException
- static java.io.OutputStream openOutputStream(String) throws java.io.IOException
- final static int READ
- final static int READ_WRITE
- final static int WRITE

interface ContentConnection implements StreamConnection

- abstract String getEncoding()
- abstract long getLength()
- abstract String getType()

- **interface Datagram implements java.io.DataInput , java.io.DataOutput**
  - abstract String getAddress()
  - abstract byte[] getData()
  - abstract int getLength()
  - abstract int getOffset()
  - abstract void reset()
  - abstract void setAddress(String) throws java.io.IOException
  - abstract void setAddress(Datagram)
  - abstract void setData(byte[],int,int)
  - abstract void setLength(int)
- **interface DatagramConnection implements Connection**
  - abstract int getMaximumLength() throws java.io.IOException
  - abstract int getNominalLength() throws java.io.IOException
  - abstract Datagram newDatagram(byte[], int) throws java.io.IOException
  - abstract Datagram newDatagram(byte[], int,String) throws java.io.IOException
  - abstract Datagram newDatagram(int) throws java.io.IOException
  - abstract Datagram newDatagram(int, String) throws java.io.IOException
  - abstract void receive(Datagram) throws java.io.IOException
  - abstract void send(Datagram) throws java.io.IOException
- **interface HttpConnection implements ContentConnection**
  - final static String GET
  - abstract long getDate() throws java.io.IOException
  - abstract long getExpiration() throws java.io.IOException
  - abstract String getFile()
  - abstract String getHeaderField(int) throws java.io.IOException
  - abstract String getHeaderField(String) throws java.io.IOException
  - abstract long getHeaderFieldDate(String, long) throws java.io.IOException
  - abstract int getHeaderFieldInt(String,int) throws java.io.IOException
  - abstract String getHeaderFieldKey(int) throws java.io.IOException
  - abstract String getHost()
  - abstract long getLastModified() throws java.io.IOException
  - abstract int getPort()
  - abstract String getProtocol()
  - abstract String getQuery()
  - abstract String getRef()
  - abstract String getRequestMethod()
  - abstract String getRequestProperty(String)
  - abstract int getResponseCode() throws java.io.IOException
  - abstract String getResponseMessage() throws java.io.IOException
  - abstract String getURL()
  - final static String HEAD
  - final static int HTTP_ACCEPTED
  - final static int HTTP_BAD_GATEWAY
  - final static int HTTP_BAD_METHOD
  - final static int HTTP_BAD_REQUEST
  - final static int HTTP_CLIENT_TIMEOUT
  - final static int HTTP_CONFLICT
  - final static int HTTP_CREATED
  - final static int HTTP_ENTITY_TOO_LARGE
  - final static int HTTP_EXPECT_FAILED
  - final static int HTTP_FORBIDDEN
  - final static int HTTP_GATEWAY_TIMEOUT
  - final static int HTTP_GONE
  - final static int HTTP_INTERNAL_ERROR
  - final static int HTTP_LENGTH_REQUIRED
  - final static int HTTP_MOVED_PERM
  - final static int HTTP_MOVED_TEMP
  - final static int HTTP_MULT_CHOICE
  - final static int HTTP_NO_CONTENT
  - final static int HTTP_NOT_ACCEPTABLE
  - final static int HTTP_NOT_AUTHORITATIVE
  - final static int HTTP_NOT_FOUND
  - final static int HTTP_NOT_IMPLEMENTED
  - final static int HTTP_NOT_MODIFIED
  - final static int HTTP_OK
  - final static int HTTP_PARTIAL
  - final static int HTTP_PAYMENT_REQUIRED
  - final static int HTTP_PRECON_FAILED
  - final static int HTTP_PROXY_AUTH
  - final static int HTTP_REQ_TOO_LONG
  - final static int HTTP_RESET
  - final static int HTTP_SEE_OTHER
  - final static int HTTP_TEMP_REDIRECT
  - final static int HTTP_UNAUTHORIZED
  - final static int HTTP_UNAVAILABLE
  - final static int HTTP_UNSUPPORTED_RANGE
  - final static int HTTP_UNSUPPORTED_TYPE
  - final static int HTTP_USE_PROXY
  - final static int HTTP_VERSION
  - final static String POST
  - abstract void setRequestMethod(String) throws java.io.IOException
  - abstract void setRequestProperty(String, String) throws java.io.IOException
- **interface InputConnection implements Connection**
  - abstract java.io.DataInputStream openDataInputStream() throws java.io.IOException
  - abstract java.io.InputStream openInputStream() throws java.io.IOException
- **interface OutputConnection implements Connection**
  - abstract java.io.DataOutputStream openDataOutputStream() throws java.io.IOException
  - abstract java.io.OutputStream openOutputStream() throws java.io.IOException
- **interface StreamConnection implements InputConnection , OutputConnection**
- **interface StreamConnectionNotifier implements Connection**
  - abstract StreamConnection acceptAndOpen() throws java.io.IOException

22.4 References

[66] *The Java Virtual Machine Specification*
Tim Lindholm and Frank Yellin, Addison Wesley, ISBN 0-201-63452-X

[67] *Downloadable Execution Environments*
http://www.osgi.org/download

[68] *J2ME, Java 2 Micro Edition*
http://java.sun.com/j2me

[69] *CDC, Connected Device Configuration*
http://java.sun.com/products/cdc

[70] *CLDC, Connected Limited Device Configuration*
http://java.sun.com/products/cldc

[71] *Foundation Profile*
http://java.sun.com/products/foundation. This external specification is © Copyright 2000 Sun Microsystems, Inc.

Recommended Section

The following section contains recommended specifications. Recommended specifications are made public to solicit feedback. Specifications in these section may be subject to changes that are not backward compatible.

23 Name-space Specification

Version 1.0

23.1 Introduction

A consistent name-space is an important component when designing and deploying a network of OSGi service platforms. Bundles and external entities need to communicate, and a name-space definition simplifies many aspects of this communication. A name-space is also an important component for communication services and their addresses.

This specification defines a federated naming scheme that outlines the rules for formatting a federated name that is used by different communication services or mapped to addressing mechanisms like Uniform Resource Locators (URLs) and Uniform Resource Names (URNs).

This specification provides only the name-space. Actual communication services and mappings are beyond the scope of this specification.

23.1.1 OSGi Name-space Essentials

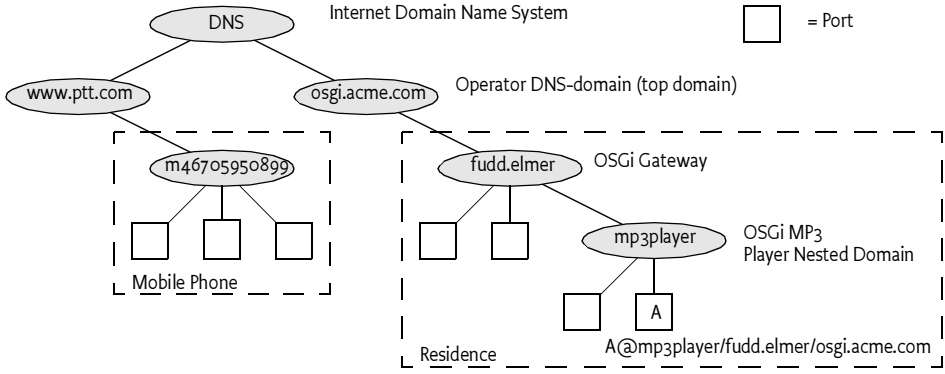
- *Powerful* – The OSGi name-space must create an environment where any bundle can send a message to any other bundle anywhere in the world. This might be restricted by available routers and security measures.
- *Flexible* – The name-space should be very flexible because the deployment models of OSGi environments differ significantly. The name-space must therefore be able to handle a large number of configurations and deployment models.
- *Non-IP compatible* – Many OSGi Service Platforms are located in systems where IP connectivity is not always present. The OSGi name-space must be able to handle addresses that are non-IP oriented.
- *Compatible with firewalls* – An OSGi Service Platform can be located behind a number of firewalls. For example, an MP3 player on an OSGi Service Platform is probably located inside a firewall implemented by a residential gateway, which is also run on an OSGi Service Platform. These gateways are further protected with an external firewall. The name-space must be able to address these nested firewalls to all levels.
- *Easy to parse* – An address in the name-space hould be easy to parse in its constituents.

23.1.2 Entities

- *Address* – The name of a location in a larger *space*. For example, an IP address is the name for a computer in cyber-space.
- *Name-space* – A set of names in which each name is unique.

- *Domain* – A domain name that can be mapped to an IP address using the Internet Domain Name Service (DNS) or other name server technology.
- *Communication Provider* – A bundle that implements a messaging or connection API for use by other bundles.
- *Top Domain* – A DNS name, like www.osgi.org, that specifies a host that controls a number of *DNS sub domains*.
- *Label* – The *label part* (defined in [73] *RFC 1035 DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*) of a DNS domain or host name. E.g. In the host name www.osgi.org, www, osgi, and org are labels. In a DNS name, it is also called a sub domain.
- *Nested domain* – A name (that may contain sub domains) that identifies a host, controlling a domain, that may reside in another domain (top or nested domain).
- *Parent domain* – The enclosing domain of a sub domain.
- *Port* – An endpoint in a communication scheme.
- *OSGi Name* – The tuple of OSGi domain, host, and port.

Figure 69 *OSGi Name-space*



23.1.3 OSGi Name Format

The format of an OSGi Name is as follows:

```
osgi-name ::= port '@' ( nested-domain '/' )* top-domain
```

The OSGi Name-space consists of a number of nested *domains*. The top domain is always an Internet Domain Name System (DNS) host name (see [72] *DNS Related RFCs* for more detail). For example, control@www.osgi.org specifies a port that is located at the OSGi web site, accessible from the Internet. The www.osgi.org is the OSGi top domain.

An OSGi Name can also contain nested domains. For example, in A@mp3player/fudd.elmer/osgi.acme.com, there are 2 nested domains, fudd.elmer and mp3player.

Both top domain, nested domains and ports may consists of *labels*. Labels are used to specify an hierarchy. Labels are separated in a name with a '.'. For example, in www.osgi.org, www, osgi and org are labels. For the top domain (which is a DNS name), a label must match a DNS sub domain. For example, in www.osgi.org, www is a DNS sub domain of osgi, and osgi is a sub domain of org. Nested domains may interpret the label in the same way but may also interpret this name in a proprietary manner.

All parts of an OSGi Name must follow the same rules for character set and comparison.

```
port            ::= name
top-domain      ::= name
nested-domain   ::= name
```

All OSGi Names strictly follow the rules of DNS.

```
name            ::= ( label '.' ) * label
label           ::= l | ( l ldh* ld )
l               ::= [A-Za-z]?
ld              ::= l | [0-9]
ldh             ::= ld | '-'
```

All names (port, top domains, and sub domains) match case insensitive when they are used in comparisons. For example, WWW.OSGI.org is the same as www.osgi.org. Such names must consist *only* of the alphabetic and numeric characters of the US-ASCII code set. A hyphen or minus sign ("–" or \u002D) is also allowed inside a name. A name must start with an alphabetic character.

In an OSGi Name, nested domains and labels can be nested to any depth. Nested domains are separated with a forward slash ('/' or \u002F). Labels are separated with a period ('.' or \u002E).

Nested domains are different from sub domains. A sub domain must use the *same* naming server technology as its parent domain. A nested domain can use a *different* naming server technology. For example, news@45705950899.mobile/osgi.acme.com must use the Internet DNS to locate the .com top level domain server. The top level domain server locates the acme domain server that resolves the address of the osgi domain server. However, the nested domain 45705950899.mobile uses the international ISDN telephony numbering scheme to locate a mobile phone.

23.1.4 Relative Addressing

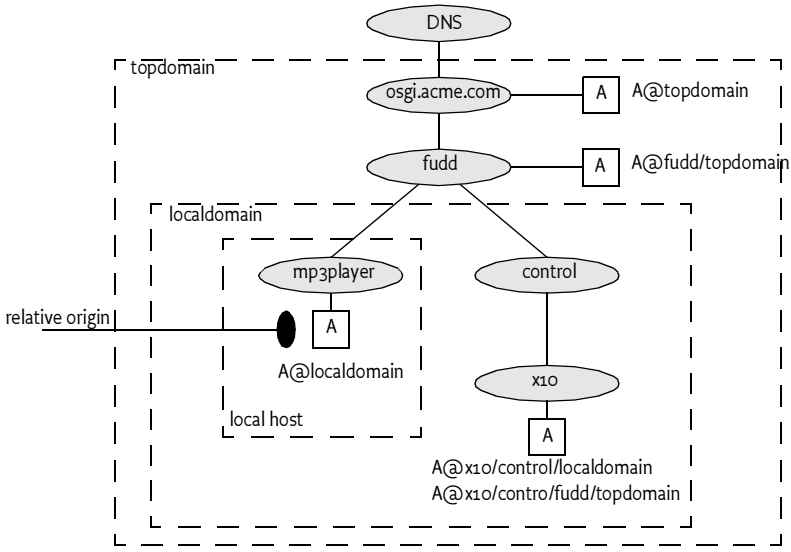
A name in this name-space can also name ports on other systems using a relative address. For example, if a bundle wants to send a message to a port on the same machine, it can use the reserved top domain name: localdomain.

For example, a@localdomain signifies a port on the local machine. The localdomain name can also be used to address machines that are located within the same domain. For example, a@x10/controller/localdomain addresses a domain controller that must be available in the same domain in which the local machine resides.

Addressing can also start from the top domain. The top domain is the host available on the Internet. The name topdomain is reserved for this domain. For example, a@x10/controller/fudd/topdomain is addressing a host relative to the top domain.

Both the localdomain name and the topdomain name are depicted in Figure 70 on page 484.

Figure 70 *Relative addressing*



Both localdomain and topdomain are reserved domain names. They must not be used as host names or domain names. A number of examples are listed in the following table.

Name	Comments
a@mp3player/localdomain	Address of a port on a host mp3player that is connected in the same domain as the originator.
a@x10/h12312/localdomain	The h12312 domain must be on the same domain as the local host. This domain needs to have a nested domain x10.
a@localdomain	Address of the port on a on the local host.
a@localdomain/www.osgi.org	*Invalid.* Localdomain must only be used as the top domain.
news@46705950899/tel/topdomain	Address relative to the top domain (assuming the OSGi would have defined the news port).

Table 27 *OSGi reserved name usage*

Name	Comments
topdomain@x10/www.acme.com	Valid, but confusing because topdomain is a reserved word for the top- and nested domains.
a@fudd/localdomain/topdomain	*Invalid.* Both localdomain and topdomain must be the last domain (right most) and can thus never be used together.

Table 27 *OSGi reserved name usage*

23.1.5 Port Names

An OSGi Name represents a path through a tree in which the domains are the nodes. Ports can reside at each node except the root. A port identifies an *endpoint*. Communication providers in the OSGi Service Platform should allow bundles to register listeners for a specific port.

A port name must follow the same rules as domain names, except that the meaning of sub domains should be reversed. The reversal is necessary to make port names compatible with standard Java practices of reversed domain names. Bundles should use unique port names that do not conflict with other bundles. This problem is usually solved using reverse domain names.

Port names have a great deal in common with the OSGi Framework's Persistent IDentities (PIDs). Whenever possible, the PIDs and port names should be aligned.

Port names without a label are to be defined only by the OSGi organization. This specification does *not* define any such names.

Temporary port names must be assigned by the communication provider. This provider must assure that those names are unique for the host and follow the rules for port names.

A number of prefixes should be used to indicate the scheme that is used to make a port name unique or indicate its purpose. The following table lists a number of recommended prefixes.

Prefix	Example(s)	Comments
com., net., org., gov., edu., biz., info., etc.	com.acme.automation	If a company has registered domain name, then the reversed domain name can be used create a PID.
guid.	guid.678A-FFA8-19AC-FF7A	The guid. prefix is reserved for automatically generated globally unique ids.

Table 28 *Port name construction examples*

Prefix	Example(s)	Comments
bundle.	bundle.16.control	Valid, using the pattern to start with the bundle ID to make the string unique
temp.	temp.com.acme.transport temp.guid.678A-FFA8-19AC-FF7A	Temporary port

Table 28 *Port name construction examples*

The following table lists a number of valid and invalid port names.

Prefix	Comments
com.acme.x10.control	Valid port name based on the reverse domain name system.
news	*Invalid* because the name does not have a sub-domain and the OSGi Specifications have not defined this port name.
bundle.67.display	Valid name based on the bundle scheme.

Table 29 *Examples of port names*

If the port name contains an hierarchy, it is advisable to separate the parts of the hierarchy with periods, similar to DNS names.

23.1.6 International Names

DNS domain names, and nested domains, cannot handle international names that contain characters not in the basic DNS character set because they must be constructed from a limited code set. This can be a limiting factor when systems are deployed in countries that use non-ASCII scripts.

Work is underway to make DNS more useful for non-english alphabets. Unfortunately, at the time of the writing of this standard, there is no consensus of how non-ASCII names should be treated in the DNS. There exist a number of proposals for encodings that can be used with DNS. Each encoding scheme is identified with a unique prefix. For example, the AMC-ACE-Z, see [82] *AMC-ACE-Z draft*, encoding should start with the prefix zq--.

It is recommended that nested domains follow the conventions used with the international names in DNS.

23.2 Related Standards

23.2.1 Uniform Resource Name (URN)

A special URI is a Uniform Resource Name (URN). A URN is a URI that is a persistent identifier for an information resource. A URN consists of the following parts:

```
urn ::= "urn:" NID ":" NSS
NID = Name-space Identifier
NSS = Name-space Specific String
```

Some examples of valid URNs:

```
urn:X-OSGi:control@467059508999%2Fmobile%2Fwww.osgi.org
urn:X-OSGi:com.acme.x@ ptt.info
```

More information about URNs can be found in [77] *Uniform Resource Name*.

23.3 Security

A name-space in itself is not related to security. Security becomes relevant when a name-space is used as an address in conjunction with a transport mechanism. Other OSGi specifications may use the OSGi Name in conjunction with transport mechanisms.

23.4 References

[72] *DNS Related RFCs*
http://www.dns.net/dnsrd/rfc

[73] *RFC 1035 DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*
http://www.ietf.org/rfc/rfc1035.txt

[74] *OSGi Service Platform, Release 2*
http://www.osgi.org/resources/spec_overview.asp

[75] *Unicode*
http://www.unicode.org

[76] *US-ASCII or basic latin*
http://www.unicode.org/charts/PDF/U0000.pdf

[77] *Uniform Resource Name*
http://www.ietf.org/html.charters/urn-charter.html

[78] *URN assigned name-spaces*
http://www.iana.org/assignments/urn-name-spaces

[79] *RFC 2141 URN Syntax*
http://www.ietf.org/rfc/rfc2141.txt

[80] *Naming and Addressing: URIs, URLs, ...*
http://www.w3.org/Addressing

[81] *International Domain names*
http://www.i-dns.net/technology/howidns/howidns.html

[82] *AMC-ACE-Z draft*
http://www.ietf.org/internet-drafts/draft-ietf-idn-amc-ace-z-01.txt
This is a draft that is likely to be replaced with an RFC in the near future.

[83] *Internationalized Domain Name Conversion Tool*
http://mct.verisign-grs.com

[84] *Preparation of Internationalized Host Names*
http://www.i-d-n.net/draft/draft-ietf-idn-nameprep-03.txt

24 Jini™ Driver Service Specification

Version 1.0

24.1 Introduction

The Jini™ network technology enables devices to form impromptu communities that can be assembled without any planning, installation, or human intervention. Each device provides services that other devices in the community can use. A unique aspect of the Jini protocols is that each participating device in the community can provide Java code to other devices so that they can leverage the provided services locally. The OSGi Service Platform is an excellent match for Jini services.

This specification outlines the rules for using Jini in an OSGi Service Platform and presents an APIs for:

- Discovery and control of Jini services within an OSGi framework
- Export of OSGi Services as Jini services.

This specification is based on the Jini specification but does not explain the Jini operations in detail. For more information on the Jini operations, read [85] *Jini*.

The OSGi Jini specification must be used with caution when used in secure systems. Jini downloads code from external devices into the OSGi Service Platform. The Jini specification has no security architecture to address the possible threats that arise out of this. This means that most code will run with permissions defined by the Jini Driver service bundle. See *Security* on page 499 for more information.

24.1.1 Essentials

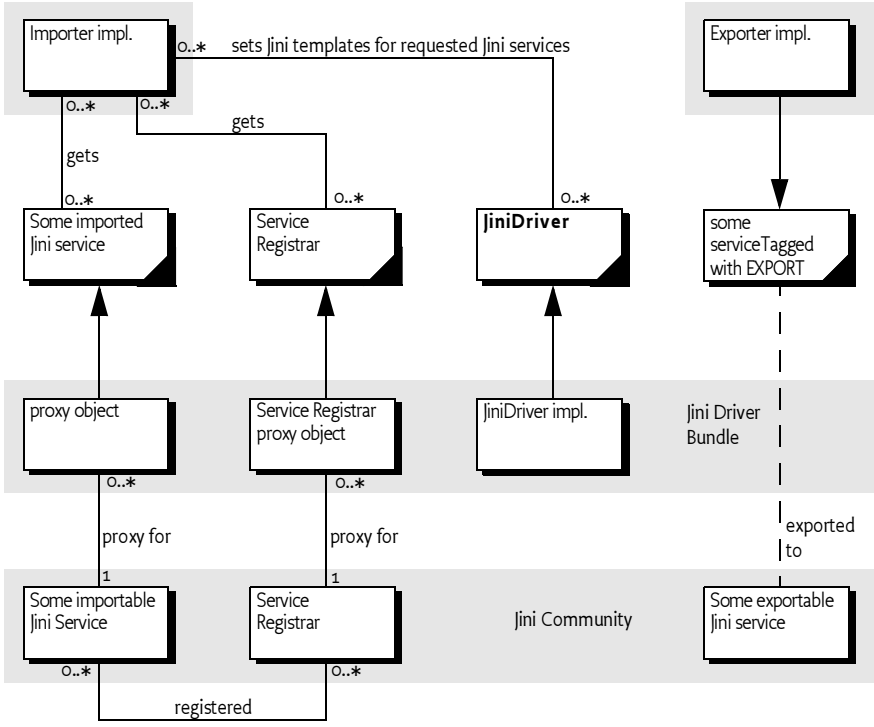
- *Transparency* – Jini services should be usable as OSGi services and vice versa.
- *Export Capability* – A bundle should be able to export an OSGi service as a Jini service.
- *Import Capability* – A bundle should be able to use an OSGi service that represents a Jini service.

24.1.2 Entities

- *Jini Service* – A Service Registrar or a service available from the Service Registrar.
- *OSGi Service* – A service object available in the OSGi service registry.
- *Group* – A group of Service Registrars.

- *Service Registrar* – The Jini service that can be discovered via a network and that provides registrations from other services.
- *Bridge* – The connection between a non-Jini device and the Jini community.
- *Proxy* –An object that acts as a stand-in for another object.
- *Jini Driver* service – The service that allows a bundle to indicate what Jini services should be registered as OSGi services.
- *Jini Service Template* – A set of net.jini.core.entry.Entry objects used to filter Jini services of interest.
- *Jini Entry* – A base class for an attribute assertion. Actually assertions like location, name, etc. are implemented in sub-classes.

Figure 71 Log Service Class Diagram org.osgi.service.jini package



24.1.3 Prerequisites

This specification requires OSGi Framework version 1.2 or higher because the Jini Driver can only be implemented with dynamic import. See *Package Management* on page 497.

24.1.4 Operation

When a Jini device is attached to a network, it uses an added protocol, called *discovery and join*, to register a service with a Jini Service Registrar. A device that provides a Jini service first locates the lookup service (*discovery*), and then uploads an object that implements all its service interfaces (*join*) via serialization.

Through the Service Registrar it is also possible to find specific services based on the value of attributes. A client can download such a service in its Virtual Machine, including the classes it needs. This service can then be executed locally in the client.

The Jini Driver must also be able to register OSGi services with a Jini registrar and must be able to import Jini services into the OSGi environment using the Jini network protocols.

24.2 The Jini Driver Service

To import services, the Jini Driver service transforms any Jini lookup service registrars that are discovered in the network, and transforms the Jini services held in the service registrars to OSGi services. The Jini Driver Service is an OSGi bundle that operates according to the driver model of the OSGi *Device Access Specification* on page 223 and that deals with Jini lookup services and OSGi services.

This specification defines a bridge between a Jini network (*community*) and an OSGi Service Platform. Using this specification, OSGi services from the service registry can be exported with very little effort to the Jini network, and Jini services from the Jini network can be imported into the OSGi framework. This is summarized as follows:

- *Exporting* – OSGi-to-Jini
- *Importing* – Jini-to-OSGi

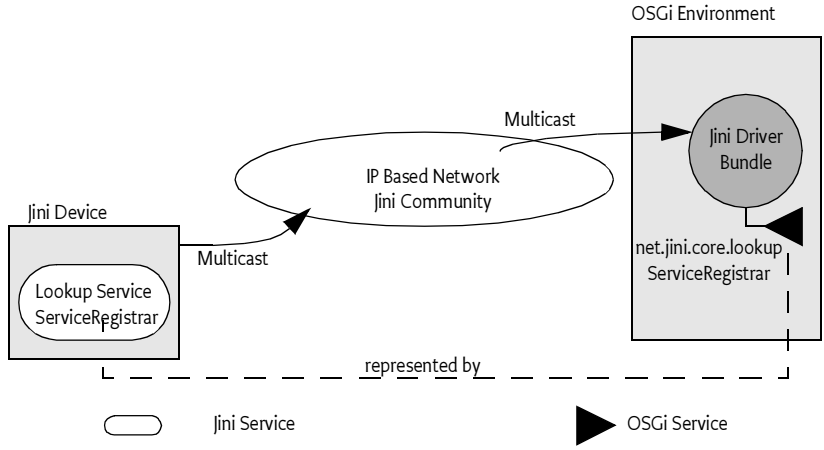
To export services, the Jini Driver can discover services that are registered in the OSGi framework as compliant with the Jini technology and register them with the discovered Jini lookup service registrars.

Using the Jini Driver bundle *significantly* simplifies the process of using or providing Jini services. Developers do not need to call the standard discovery and join mechanisms because the Jini Driver service does this for them. From a developer's point of view, there is no difference between Jini services and conventional OSGi services. Any operation defined in OSGi can be performed on Jini services.

24.3 Discovering Services

When started, the Jini Driver service must listen to the network for Jini Lookup discovery events using the standard Jini discovery classes as defined in the Jini specifications.

Figure 72 The Jini Driver multicast discovery of registrars



When a Jini Lookup Service Registrar is discovered on the network, the Jini Driver must retrieve its *proxy object* and register it as a net.jini.core.lookup.ServiceRegistrar service with the OSGi Framework. These Service Registrars can be used by bundles that are written to work directly with the Jini protocol.

The Jini Driver must then fetch all services from this Service Registrar that match the Jini net.jini.core.lookup.ServiceTemplate objects that are registered by Jini aware bundles with the setServiceTemplates(net.jini.core.lookup.ServiceTemplate[]) method.

Each of these services must be registered in the Framework's service registry under the interfaces or classes that are specified in the associated net.jini.core.lookup.ServiceTemplate object (a ServiceTemplate object filters the Jini services of interest). Additionally, the following registration properties should be set when available:

- ENTRIES – Must contain an array of net.jini.core.entry.Entry objects (Entry[]). These are the entries that are associated with the attributeSets of the net.jini.core.lookup.ServiceItem object in the lookup.
- SERVICE_ID – This property is a unique identity for a Jini service. The type is a String.

24.3.1 Finding a Jini Service in the OSGi Service Registry

Typically in Jini, it is necessary to construct a net.jini.core.lookup.ServiceTemplate object with a number of net.jini.core.entry.Entry classes. The net.jini.core.lookup.ServiceTemplate object can do an assertion on the values of service attributes. These attributes are not available in the OSGi service registry because Jini does not provide a defined way to translate an net.jini.core.entry.Entry object into a string or number that can be matched with an OSGi filter.

Jini specifically keeps the full semantics of the assertion in an net.jini.core.entry.Entry sub-class. This means that *only* equality assertions are possible. Jini does not support assertions such as greater than, less than, presence, or other filter operations like OSGi does (or Service Location Protocol, SLP). For example, in Jini it is impossible to find a printer that has more than 100 pages available, only an exact match is possible.

It is thus necessary to post-filter the services when it is required to look for services with specific Jini attributes. This is shown in the following example.

```
import org.osgi.framework.*;
import net.jini.lookup.entry.Location;

interface Display { void show(String text); }

class Publish {
   ...
   ServiceReference SearchInFramework(
      BundleContext bc, Entry match ) {
      try {
         String filter =
            "(objectClass=" + Display.class.getName() + ")";
         Entry entry = new Location(
            "home","ground-floor","living" );
         ServiceReference[] ref = bc.getServiceReferences(
            null, filter);
         for (int i=0; ref!=null && i<ref.length; i++) {
            Entry[] entries = (Entry[])
               ref[i].getProperty(JiniDriver.ENTRIES);
            for ( int j=0;
               entries!=null && j<entries.length; j++ )
                  if ( entry.equals(match))
                     return ref[i];
         }
      } catch (InvalidSyntaxException ex) {
         ...
      }
      return null;
   }
   ...
}
```

24.3.2 Using the Jini Service Registrar

Alternatively, a client bundle can use the Service Registrar objects that are also available in the OSGi service registry. The ServiceRegistrar interface takes a net.jini.core.lookup.ServiceTemplate object as a parameter and filters accordingly.

The following example demonstrates how Jini services can be found using the Jini Service Registrar services found in the OSGi framework.

```
import net.jini.core.lookup.*;
import net.jini.core.entry.*;
```

```
import net.jini.lookup.entry.Location;
import org.osgi.framework.*;
import java.rmi.RemoteException;

interface DisplayInterface { void show(String text); }

class Publish {
 Object searchInJini(ServiceRegistrar lus) {
     try {
        Class[] classes = { DisplayInterface.class };
        Entry[] entries = { new Location(
           "home", "ground-floor", "living" )};

        ServiceTemplate template =
           new ServiceTemplate(null, classes, entries);

        return lus.lookup(template);
     } catch (RemoteException ex) {
        ...
     }
     return null;
  }
}
```

24.4 Importing a Jini Service

The Jini-to-OSGi import capability enables applications in an OSGi Framework to interact with Jini services. Thus, OSGi bundles need not include extra components to use Jini services, and they do not even have to be aware of the the fact that the Service Platform is Jini enabled.

However, if an OSGi bundle needs access to specific services, it must fetch the Jini Driver service and register an array of net.jini.core.lookup.ServiceTemplate objects with the setServiceTemplates(net.jini.core.lookup.ServiceTemplate[]) method. A net.jini.core.lookup.ServiceTemplate object acts as a filter for Jini services of interest.

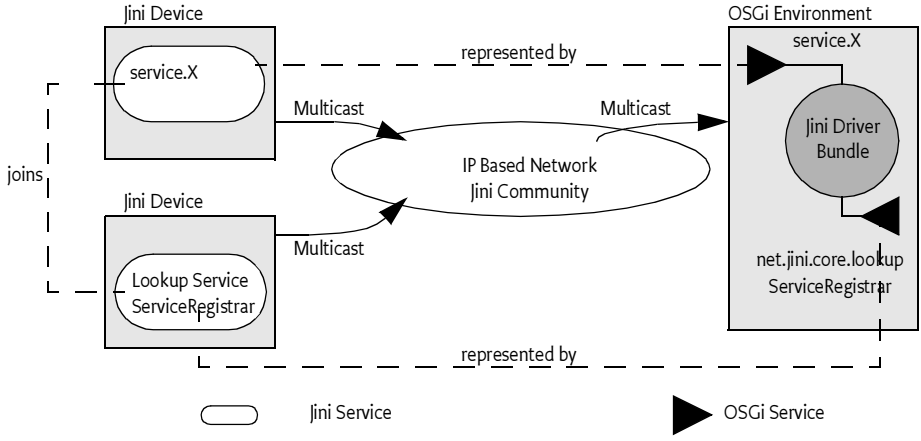
Each bundle can set its own service templates that match Jini services. All set templates must be merged by the Jini Driver server. *All* Jini services that match the current set of templates must be imported by the Jini Driver service.

```
public class Pub extends ServiceTracker {
   ServiceTemplate      templates[];

   Pub( BundleContext context ) {
      super( context,
         JiniDriver.class.getName(), null );
      Entry entries[] = {
         new Location( "stuga","ground","main" ) };
      Class classes[] = { Display.class };
      ServiceTemplate templates[] = {
```

```
            new ServiceTemplate(
               null, classes, entries ) };
         open();
      }

      public Object addingService( ServiceReference ref ) {
         JiniDriver jd = (JiniDriver) super.addingService(ref);
         jd.setServiceTemplates( templates );
      }
   }
```

The registration of the discovered Jini services is only possible if the interfaces under which they are registered are available to the OSGi Framework. Thus, the packages of these interfaces must be exported by a bundle in the OSGi Service Platform. See *Package Management* on page 497 for more information about this subject.

Figure 73 *Importing Jini Services*



When a Jini service is imported, it is registered as an OSGi service with the following set of properties:

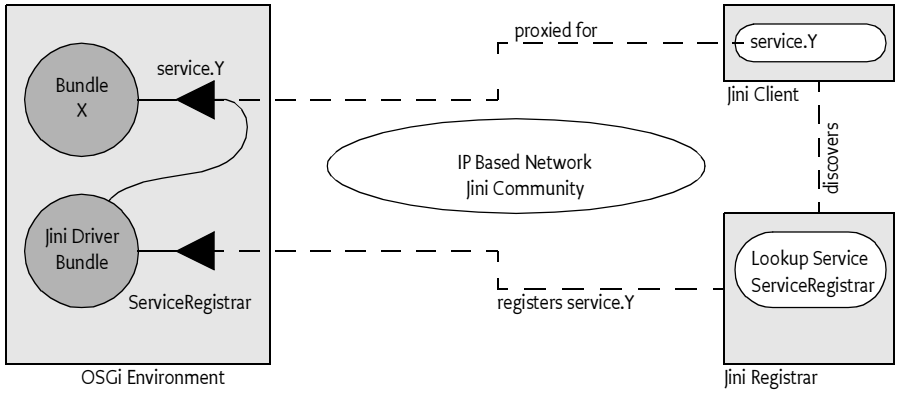
- DEVICE_CATEGORY – The property that must be used in order to participate in the OSGi Device Access mechanisms. The Jini Driver must set the value of this property to jini when it imports a Jini service.
- SERVICE_ID – A unique Jini service id. Each service is required to have a unique service id. The Jini Driver sets this property to the id of the service in the Service Registrar.
- ENTRIES – The entries that describe the service to Jini. The type of this property must be net.jini.core.entry.Entry[]. These Entry objects are copied from the net.jini.core.entry.Entry attributes in the Service Registrar.

24.5 Exporting an OSGi Service to Jini

A Jini Driver must register an OSGi service with all discovered Jini registrars when the service is *Jini compliant* and registered with a registration property JiniDriver.EXPORT.

A Jini Driver must manage the life-cycle of Jini services according to the life-cycle state of the related OSGi service. I.e., when the service is registered with the framework, the Jini Driver must automatically join this OSGi service with the discovered Jini Lookup Services. When the OSGi service becomes unregistered, the Jini Driver must remove it from all the Jini Service Registrars. This must also cancel the associated Jini leases.

Figure 74 *Exporting Jini Services*



A Jini service is a conventional OSGi service with some additional requirements from the Jini specification:

- The exported service object, and all the objects it directly refers to, *must* be serializable. Some objects, such as Swing's JTextArea, cannot currently be serialized and therefore cannot be used.
- The exported OSGi service object is created in the local Virtual Machine (VM), but when it runs it does so in the client's VM. This typically means it needs to be a proxy for the actual service that is available in the OSGi Service Platform.

An exported service must be registered with the following properties:

- DEVICE_CATEGORY – The device category property as defined in the *Device Access Specification* on page 223, service specification. The value of this property must be "jini". This property must be set.
- EXPORT – Indicates to the Jini Driver service that this service wants to be exported as a Jini service. The value is irrelevant, the Jini Driver service must use the presence operator (=*) to detect this property. This property must be set.
- LUS_EXPORT_GROUPS – The names of the Jini groups in which this exported service wants to participate. The type is an array of String objects (String[]). If this property is not set, the Jini Driver default is used. This property is optional.

- SERVICE_ID – A unique Jini service id set by the exporting bundle. This id is used to identify the service in the Service Registrar. It must be a String object. If it is not set, the Jini Driver must create a unique id. This property is optional.
- ENTRIES – An array of net.jini.core.entry.Entry objects. These are the attributes describing the service and are normally used to find Jini services of interest. This property is optional.

24.5.1 Example

```
void foo(BundleContext context) {
   String [] groups = { "acme", "osgi" };
   Entry [] entries = {
      new Location( "hut", "attic", "back" ) };
   Hashtable ht = new Hashtable();
   ht.put( JiniDriver.EXPORT, "" );
   ht.put( JiniDriver.LUS_EXPORT_GROUPS, groups );
   ht.put(
      org.osgi.service.device.Constants.DEVICE_CATEGORY,
      JiniDriver.DEVICE_CATEGORY );
   ht.put( JiniDriver.ENTRIES, entries );
   ht.put( Constants.SERVICE_PID, getPid() );
   context.registerService( Foo.class.getName(), this, ht );
}
```

24.6 Package Management

The Jini Driver loads code from other VMs and executes this code in the OSGi environment. The strict management of classloader management in the OSGi Service Platform affects how Jini applications can be used. The following sections discuss the different package related issues that occur when the Jini protocol is used with an OSGi Service Platform.

24.6.1 Jini Service Interfaces

The OSGi Framework registers service objects under the name of an interface. This interface must be exported by a bundle before it can be used by another bundle. Each registered Jini service interface must therefore be exported by one or more bundles if it is to be useful.

This implies that the Jini Driver must assure that the Jini service object implements the actual service interface (the identical class object) that was exported in the OSGi Service Platform. The Jini Driver cannot foresee what Jini Services it must support a priori, and is thus not able to export or import all the possible packages. Therefore, the Jini Driver requires dynamic import and can be implemented only on Service Platforms with a Framework version 1.2 or higher.

24.6.2 Java RMI Package

Jini requires a number of Remote Method Invocation (RMI) classes but can be implemented with other communication schemes. The RMI requirement comes from the fact that the wire protocol depends on some pivotal RMI classes. These are:

- java.rmi.MarshalException
- java.rmi.MarshalledObject
- java.rmi.NoSuchObjectExceptionRemote
- java.rmi.RemoteException
- java.rmi.UnmarshalException

Further RMI dependencies are related to the Jini implementation. The Sun Jini Reference Implementation requires full RMI Support and a Java 2 compatible VM. None of the OSGi Execution Environments contain these classes.

24.6.3 Jini Packages

The Jini specifications define many Java APIs that are needed to implement a Jini Driver for an OSGi Service Platform. The Jini Driver bundle should export the following packages:

Package	Description
net.jini.core.entry	Entry Specification
net.jini.core.lookup	Lookup Service Specification
net.jini.core.event	Distributed Event Specification
net.jini.core.lease	Distributed Leasing Specification
net.jini.core.discovery	Unicast discovery part of the Jini Discovery and Join Specification
net.jini.admin	The interface that an administrable Jini service should implement
net.jini.discovery	Discovery Utilities Specification
net.jini.lookup	Utilities for managing the communication with Jini Lookup Services
net.jini.lookup.entry	Utility implemented in order to specify a modification of an attribute entry
net.jini.lease	Utility and event classes for leasing

Table 30 *Jini exported packages*

24.7 Configuration

The Jini Driver service should use the OSGi Configuration Admin service (*Configuration Admin Service Specification* on page 181) to specify the Jini groups to which OSGi services should be exported and the Jini groups that should be imported.

However, this specification also defines a number of System properties for this purpose. The Jini Driver services define two configuration properties:

- CM_LUS_EXPORT_GROUPS – An array of String objects that specifies the groups with which an OSGi service marked with EXPORT should be registered. If this array is not set, the OSGI service must be registered with all groups.
- CM_LUS_IMPORT_GROUPS – An array of String objects that specifies the groups that should be imported. The array defines the groups of Service Registrars in which the driver is interested. If it this array is not set, it must default to all groups.

These properties can also be set in the System properties, which will then override configuration properties. Multiple elements in these properties must be separated by commas.

24.8 Security

The Jini concept requires the device to download Java code from a peer device. This code is then executed in the device's own VM. This is a great threat to security. Worms and viruses can easily spread this way. In addition, Jini does not have a security architecture.

The Jini Driver implementation must therefore exercise extreme care not to allow the Jini services any permissions that can allow them to do harm.

Jini services inherit the permissions of the Jini Driver service bundle. This bundle needs access to the local network in order to participate in the Jini community. Jini services also usually require access to the local network because they often are a proxy for a remote service.

24.9 org.osgi.service.jini

The OSGi Jini Driver. 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.jini; specification-version=1.0
```

24.9.1 public interface JiniDriver

A basic interface for a Jini Driver.

This Driver acts as a bridge between a Jini network (community) and an OSGi framework. Using this driver, OSGi services can be exported to the Jini network, and Jini services from the Jini network can be imported into the OSGi framework. This results in two possible transformations: Jini-to-OSGi and OSGi-to-Jini. The Jini Driver is responsible for these transformations.

An OSGi service is a Jini service if it is registered in the framework with the specified properties.

In OSGi-to-Jini transformation, the driver registers OSGi services as Jini services in the discovered LUS.

In Jini-to-OSGi transformation it registers with the framework all discovered LUS and services in the LUS matching the given template.

The Jini Driver can be configured, through a set of properties, to export/import Jini Services.

The properties DEVICE_CATEGORY, EXPORT, LUS_EXPORT_GROUPS, SERVICE_ID, and ENTRIES are service register properties for particular Jini Service (imported or exported).

The properties CM_LUS_IMPORT_GROUPS and CM_LUS_EXPORT_GROUPS are for configuration of the Jini Driver. These properties are kept in the Configuration Management Service defined by OSGi.

24.9.1.1 public static final String CM_LUS_EXPORT_GROUPS = "jini.lus.export.groups"

Optional service property, which should be a string array, containing the LUS groups that the driver is interested in, when exporting framework services to the Jini network. The driver discovers only the LUS, which are members of at least one of these groups. It discovers all if the property is null or the property is not defined, and does not perform discovery if the length of the array is 0. If Jini Lookup Services are discovered, which after changing the value of this property are not members of the groups, the registration of all Jini services from the framework, which are registered with them, is cancelled. The name of the property is `jini.lus.export.groups`.

24.9.1.2 public static final String CM_LUS_IMPORT_GROUPS = "jini.lus.import.groups"

Optional service property, which should be a string array, containing the groups of LUS, that the driver is interested in, when importing Jini services. The driver discovers only the LUS members of at least one of these groups. It discovers all if the property is null or the property is not defined, and does not perform discovery if the length of the array is zero. If LUS are discovered, which after changing the value of this property are not members of the groups, all registered services from them are unregistered. The name of the property is `jini.lus.import.groups`.

24.9.1.3 public static final String DEVICE_CATEGORY = "jini"

Constant for the value of the service property DEVICE_CATEGORY used by all Jini services.

Value: `jini`

See Also `org.osgi.service.device.Constants.DEVICE_CATEGORY`

24.9.1.4 public static final String ENTRIES = "jini.entries"

Optional service property, which should be an net.jini.core.entry.Entry array, holding the attributes set of the framework service that represents Jini proxy in the registration with a LUS. The name of the property is `jini.entries`.

See Also `net.jini.core.entry.Entry`

24.9.1.5 public static final String EXPORT = "jini.export"

The Export service property is a hint that marks an OSGi service to be picked up and exported by the Jini Driver in the Jini network. Imported services must not have this property set.

The property has no value. The name of the property is jini.export

24.9.1.6 public static final String LUS_EXPORT_GROUPS = "jini.lus.export.groups"

Optional service property, which should contain a string array of the LUS groups that are of interest to the OSGi service. This overrides the property CM_LUS_EXPORT_GROUPS of the Jini Driver. If the value of this property is not defined, CM_LUS_EXPORT_GROUPS will be used. The name of the property is jini.lus.export.groups.

24.9.1.7 public static final String SERVICE_ID = "jini.service.id"

Optional service property, which should contain a string representation of the Jini service ID. It is used by the Jini Driver when exporting framework service. The driver automatically fills the values of this property when importing the Jini service. The name of the property is jini.service.id.

24.9.1.8 public ServiceTemplate[] getServiceTemplates( )

- Gets the current set of templates that is used for searching and registering services registered in discovered LUS. A service, registered in a LUS, will be registered in the framework if it matches at least one of the templates in this set

Returns an array containing templates or null if the set of templates is empty.

24.9.1.9 public void setServiceTemplates(ServiceTemplate[] template)

template template to be added.

- The Jini Driver is defined as a Service Factory. For every bundle a different set of Service Templates is maintained. This method sets a new set of ServiceTemplates (net.jini.core.lookup.ServiceTemplates) that are used for searching and registering services in the discovered LUS. A service registered in a LUS will be registered in framework if it matches at least one of the templates in one of the sets.

  The ServiceTemplate(null, null, null) matches all services.

24.10 References

[85] *Jini*
http://www.jini.org

25 UPnP™ Device Service Specification

Version 1.0

25.1 Introduction

The UPnP Device Architecture specification provides the protocols for a peer-to-peer network. It specifies how to join a network and how devices can be controlled using XML messages sent over HTTP. The UPnP specifications leverage Internet protocols, including IP, TCP, UDP, HTTP, and XML. The OSGi specifications address how code can be download and managed in a remote system. Both standards are therefore fully complimentary. Using an OSGi Service Platform to work with UPnP enabled devices is therefore a very succesful combination.

This specification specifies how OSGi bundles can be developed that interoperate with UPnP™ (Universal Plug and Play) devices and UPnP control points. The specification is based on [86] *UPnP Device Architecture* and does not further explain the UPnP specifications. The UPnP specifications are maintained by [87] *UPnP Forum*.

UPnP is a trademark of the UPnP Implementers Corporation.

25.1.1 Essentials

- *Scope* – This specification is limited to device control aspects of the UPnP specifications. Aspects concerning the TCP/IP layer, like DHCP and limited TTL, are not addressed.
- *Transparency* – OSGi services should be made available to networks with UPnP enabled devices in a transparent way.
- *Network Selection* – It must be possible to restrict the use of the UPnP protocols to a selection of the connected networks. For example, in certain cases OSGi services that are UPnP enabled should not be publishedto the Wide Area Network side of a gateway, nor should UPnP devices be detected on this WAN.
- *Event handling* – Bundles must be able to listen to UPnP events.
- *Export OSGi services as UPnP devices* – Enable bundles that make a service available to UPnP control points.
- *Implement UPnP Control Points* – Enable bundles that control UPnP devices.

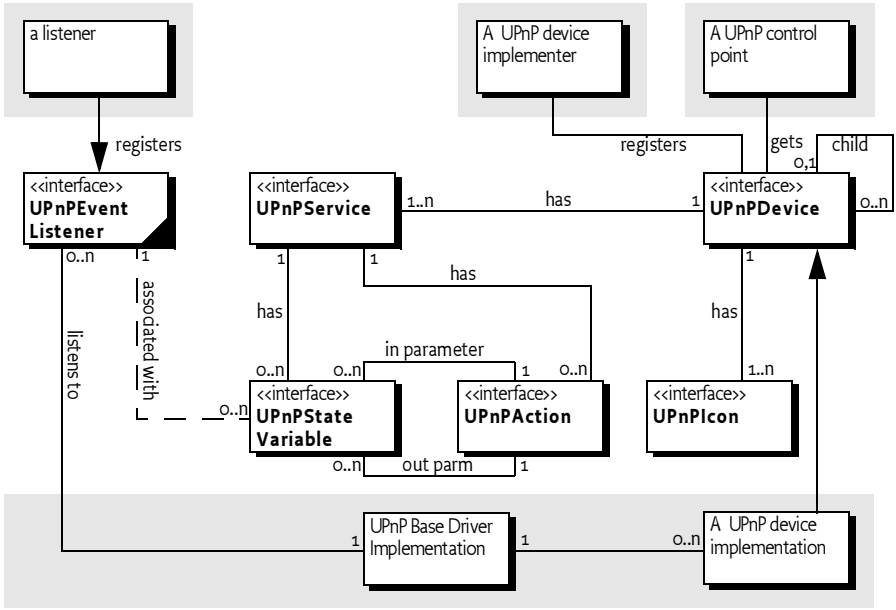
25.1.2 Entities

- *UPnP Base Driver* – The bundle that implements the bridge between OSGi and UPnP networks. This entity is not represented as a service.
- *UPnP RootDevice* –A physical device can contain one or more root devices. Root devices contain one ore more devices. A root device is mod-

elled with a UPnPDevice object, there is no separate interface defined for root devices.

- *UPnP Device* – The representation of a UPnP device. A UPnP device may contain other UPnP devices and UPnP services. This entity is represented by a UPnPDevice object.
- *UPnP Service* –A UPnP device consists of a number of services. A UPnP service has a number of UPnP state variables that can be queried and modified with actions. This concept is represented by a UPnPService object.
- *UPnP Action* – A UPnP service is associated with a number of actions that can be performed on that service and that may modify the UPnP state variables. This entity is represented by a UPnPAction object.
- *UPnP State Variable* – A variable associated with a UPnP service, represented by a UPnPStateVariable object.
- *UPnP Event Listener Service* – A listener to events coming from UPnP devices.
- *UPnP Host* – The machine that hosts the code to run a UPnP device or control point.
- *UPnP Control Point* – A UPnP device that is intended to control UPnP devices over a network. For example, a UPnP remote controller.
- *UPnP Icon* – A representation class for an icon associated with a UPnP device.
- *UDN* – Unique Device Name, a name that uniquely identifies the a specific device.

Figure 75 UPnP Service Specification class Diagram org.osgi.service.upnp package



25.1.3 Operation Summary

To make a UPnP service available to UPnP control points on a network, an OSGi service object must be registered under the UPnPDevice interface with the Framework. The UPnP driver bundle must detect these UPnP Device services and must make them available to the network as UPnP devices using the UPnP protocol.

UPnP devices detected on the local network must be detcted and automatically registered under the UPnPDevice interface with the Framework by the UPnP driver implementation bundle.

A bundle that wants to control UPnP devices, for example to implement a UPnP control point, should track UPnP Device services in the OSGi service registry and control them appropriately. Such bundles should not distinguish between resident or remote UPnP Device services.

25.2 UPnP Specifications

The UPnP DA is intended to be used in a broad range of device from the computing (PCs printers), consumer electronics (DVD, TV, radio), communication (phones) to home automation (lighting control, security) and home appliances (refridgerators, coffeemakers) domains.

For example, a UPnP TV might announce its existence on a network by broadcasting a message. A UPnP control point on that network can then discover this TV by listening to those announce messages. The UPnP specifications allow the control point to retrieve information about the user interface of the TV. This information can then be used to allow the end user to control the remote TV from the control point, for example turn it on or change the channels.

The UPnP specification supports the following features:

- *Detect and control a UPnP standardized device.* In this case the control point and the remote device share a priori knowledge about how the device should be controlled. The UPnP Forum intends to define a large number of these standardized devices.
- *Use a user interface description.* A UPnP control point receives enough information about a device and its services to automatically build a user interface for it.
- *Programmatic Control.* A program can directly control a UPnP device without a user interface. This control can be based on detected information about the device or through a priori knowledge of the device type.
- *Allows the user to browse a web page supplied by the device.* This web page contains a user interface for the device that be directly manipulated by the user. However, this option is not well defined in the UPnP Device Architecture specification and is not tested for compliance.

The UPnP Device Architecture specification and the OSGi Service Platform provide *complementary* functionality. The UPnP Device Architecture specification is a data communication protocol that does not specify where and how programs execute. That choice is made by the implementations. In contrast, the OSGi Service Platform specifies a (managed) execution point and

does not define what protocols or media are supported. The UPnP specification and the OSGi specifications are fully complementary and do not overlap.

From the OSGi perspective, the UPnP specification is a communication protocol that can be implemented by one or more bundles. This specification therefore defines the following:

- How an OSGi bundle can implement a service that is exported to the network via the UPnP protocols.
- How to find and control services that are available on the local network.

The UPnP specifications related to the assignment of IP addresses to new devices on the network or auto-IP self configuration should be handled at the operating system level. Such functions are outside the scope of this specification.

25.2.1 UPnP Base Driver

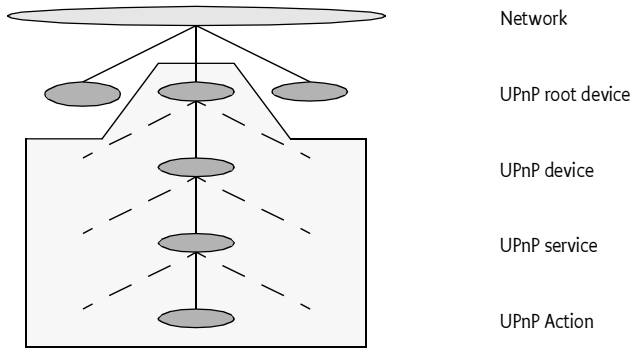
The functionality of the UPnP service is implemented in a UPnP *base driver*. This is a bundle that implements the UPnP protocols and handles the interaction with bundles that use the UPnP devices. A UPnP base driver bundle must provide the following functions:

- Discover UPnP devices on the network and map each discovered device into an OSGi registered UPnP Device service.
- Present UPnP marked services that are registered with the OSGi Framework on one or more networks to be used by other computers.

25.3 UPnP Device

The principle entity of the UPnP specification is the UPnP device. There is a UPnP *root device* that represents a physical appliance, such as a complete TV. The root device contains a number of sub-devices. These might be the tuner, the monitor, and the sound system. Each sub-device is further composed of a number of UPnP services. A UPnP service represents some functional unit in a device. For example, in a TV tuner it can represent the TV channel selector. Figure 76 on page 506 illustrates this hierarchy.

Figure 76 *UPnP device hierarchy*



Each UPnP service can be manipulated with a number of UPnP actions. UPnP actions can modify the state of a UPnP state variable that is associated with a service. For example, in a TV there might be a state variable *volume*. There are then actions to set the volume, to increase the volume, and to decrease the volume.

25.3.1 Root Device

The UPnP root device is registered as a UPnP Device service with the Framework, as well as all its sub-devices. Most applications will work with sub-devices, and, as a result, the children of the root device are registered under the UPnPDevice interface.

UPnP device properties are defined per sub-device in the UPnP specification. These properties must be registered with the OSGi Framework service registry so they are searchable.

Bundles that want to handle the UPnP device hierarchy can use the registered service properties to find the parent of a device (which is another registered UPnPDevice).

The following service registration properties can be used to discover this hierarchy:

- PARENT_UDN – The Universal Device Name (UDN) of the parent device. A root device most not have this property registered. Type is a String object.
- CHILDREN_UDN – An array of UDNs of this device's children. Type is a String[] object.

25.3.2 Exported Versus Imported Devices

Both imported (from the network to the OSGi service registry) and exported (from the service registry to the network) UPnPDevice services must have the same representation in the OSGi Service Platform for identical devices. For example, if an OSGi UPnP Device service is exported as a UPnP device from an OSGi Service Platform to the network, and it is imported into another OSGi Service Platform, the object representation should be equal. Application bundles should therefore be able to interact with imported and exported forms of the UPnP device in the same manner.

Imported and exported UPnP devices differ only by two marker properties that can be added to the service registration. One marker, DEVICE_CATEGORY, should typically be set only on imported devices. By not setting DEVICE_CATEGORY on internal UPnP devices, the Device Manager does not try to refine these devices (See the *Device Access Specification* on page 223 for more information about the Device Manager). If the device service does not implement the Device interface and does not have the DEVICE_CATEGORY property set, it is not considered a *device* according to the Device Access Specification.

The other marker, UPNP_EXPORT, should only be set on internally created devices that the bundle developer wants to export. By not setting UPNP_EXPORT on registered UPnP Device services, the UPnP Device service can be used by internally created devices that should not be exported to the network. This allows UPnP devices to be simulated within an OSGi Service Platform without announcing all of these devices to any networks.

25.3.3 Icons

A UPnP device can optionally support an icon. The purpose of this icon is to identify the device on a UPnP control point. UPnP control points can be implemented in large computers like PC's or simple devices like a remote control. However, the graphic requirements for these UPnP devices differ tremendously. The device can, therefore, export a number of icons of different size and depth.

In the UPnP specifications, an icon is represented by a URL that typically refers to the device itself. In this specification, a list of icons is available from the UPnP Device service.

In order to obtain localized icons, the method getIcons(String) can be used to obtain different versions. If the locale specified is a null argument, then the call returns the icons of the default locale of the called device (not the default locale of the UPnP control point).When a bundle wants to access the icon of an imported UPnP device, the UPnP driver gets the data and presents it to the application through an input stream.

A bundle that needs to export a UPnP Device service with one ore more icons must provide an implementation of the UPnPIcon interface. This implementation must provide an InputStream object to the actual icon data. The UPnP driver bundle must then register this icon with an HTTP server and include the URL to the icon with the UPnP device data at the appropriate place.

25.4 Device Category

UPnP Device services are devices in the context of the Device Manager. This means that these services need to register with a number of properties to participate in driver refinement. The value for UPnP devices is defined in the UPnPDevice constant DEVICE_CATEGORY. The value is UPnP. The UPnPDevice interface contains a number of constants for matching values. Refer to *MATCH_GENERIC* on page 515 for further information.

25.5 UPnPService

A UPnP Device contains a number of UPnPService objects. UPnPService objects combine actions and state variables.

25.5.1 State Variables

The UPnPStateVariable interface encapsulates the properties of a UPnP state variable. In addition to the properties defined by the UPnP specification, a state variable is also mapped to a Java data type. The Java data type is used when an event is generated for this state variable and when an action is performed containing arguments related to this state variable. There must be a strict correspondence between the UPnP data type and the Java data type so that bundles using a particular UPnP device profile can predict the precise Java data type.

The function QueryStateVariable defined in the UPnP specification has been deprecated and is therefore not implemented. It is recommended to use the UPnP event mechanism to track UPnP state variables.

25.6 Working With a UPnP Device

The UPnP driver must register all discovered UPnP devices in the local networks. These devices are registered under a UPnPDevice interface with the OSGi Framework.

Using a remote UPnP device thus involves tracking UPnP Device services in the OSGi service registry. The following code illustrates how this can be done. The sample Controller class extends the ServiceTracker class so that it can track all UPnP Device services and add them to a user interface, such as a remote controller application.

```
class Controller extends ServiceTracker {
   UI      ui;

   Controller( BundleContext context ) {
      super( context, UPnPDevice.class.getName(), null );
   }
   public Object addingService( ServiceReference ref ) {
      UPnPDevice dev = (UPnPDevice)super.addingService(ref);
      ui.addDevice( dev );
      return dev;
   }
   public void removedService( ServiceReference ref,
      Object dev ) {
      ui.removeDevice( (UPnPDevice) dev );
   }
   ...
}
```

25.7 Implementing a UPnP Device

OSGi services can also be exported as UPnP devices to the local networks, in a way that is transparent to typical UPnP devices. This allows developers to bridge legacy devices to UPnP networks. A bundle should perform the following to export an OSGi service as a UPnP device:

- Register an UPnP Device service with the registration property UPNP_EXPORT.
- Use the registration property PRESENTATION_URL to provide the presentation page. The service implementer must register its own servlet with the Http Service to serve out this interface. This URL must point to that servlet.

There can be multiple UPnP root devices hosted by one OSGi platform. The relationship between the UPnP devices and the OSGi platform is defined by the PARENT_UDN and CHILDREN_UDN service properties. The bundle registering those device services must make sure these properties are set accordingly.

25.8 Event API

UPnP events are sent using the whiteboard model, in which a bundle interested in receiving the UPnP events registers an object implementing the UPnPEventListenerService interface. A filter can be set to limit the events for which a bundle is notified.

If a service is registered with a property named upnp.filter with the value of an instance of an Filter object, the listener is only notified for matching events (This is a Filter object and not a String object because it allows the InvalidSyntaxException to be thrown in the client and not the UPnP driver bundle).

The filter might refer to any valid combination of the following pseudo properties for event filtering:

- UPnPDevice.UDN – (UPnP.device.UDN) Only events generated by services contained in the specific device are delivered. For example: (UPnP.device.UDN=uuid:Upnp-TVEmulator-1_0-1234567890001)
- UPnPDevice.TYPE – (UPnP.device.type) Only events generated by services contained in a device of the given type are delivered. For example: (UPnP.device.type=urn:schemas-upnp-org:device:tvdevice:1)
- UPnPService.ID – (UPnP.service.id) Service identity. Only events generated by services matching the given service ID are delivered.
- UPnPService.TYPE – (UPnP.service.type) Only events generated by services of of the given type are delivered.

If an event is generated, the notifyUPnPEvent(String,String,Dictionary) method is called on all registered UPnPEventListener services for which the optional filter matches for that event. If no filter is specified, all events must be delivered. If the filter does not match, the UPnP driver must not call the UPnP Event Listener service.

One or multiple events are passed as parameters to the notifyUPnPEvent(String,String,Dictionary) method. The Dictionary object holds a pair of UpnPStateVariable objects that triggered the event and an Object for the new value of the state variable.

25.8.1 Initial Event Delivery

Special care must be taken with the initial subscription to events. According to the UPnP specification, when a client subscribes for notification of events for the first time, the device sends out a number of events for each state variable, indicating the current status of each state variable. This behavior simplifies the synchronization of a device and an event-driven client.

The UPnP Driver must mimic this event distribution for all UPnP Event Listener services when they are registered. The driver must guarantee the same behavior for all registrations by keeping an internal history of the events.

The call to the listener's notification method must be done asynchronously.

25.9 Localization

All values of the UPnP properties are obtained from the device using the device's default locale. If an application wants to query a set of localized property values, it has to use the method getDescriptions(String). For localized versions of the icons, the method getIcons(String) is to be used.

25.10 Dates and Times

The UPnP specification uses different types for date and time concepts. An overview of these types is given in Table 31 on page 511.

UPnP Type	Class	Example	Value (TZ=CEST= +0200)
date	Date	1985-04-12	Sun April 12 00:00:00 CEST 1985
dateTime	Date	1985-04-12T10:15:30	Sun April 12 10:15:30 CEST 1985
dateTime.tz	Date	1985-04-12T10:15:30+0400	Sun April 12 08:15:30 CEST 1985
time	Long	23:20:50	84.050.000 (ms)
time.tz	Long	23:20:50+0300	1.250.000 (ms)

Table 31 *Mapping UPnP Date/Time types to Java*

The UPnP specification points to [91] *XML Schema*. In this standard, [92] *ISo 8601 Date And Time formats* are referenced. The mapping is not completely defined which means that the this OSGi UPnP specification defines a complete mapping to Java classes. The UPnP types date, dateTime and dateTime.tz are represented as a Date object. For the date type, the hours, minutes and seconds must all be zero.

The UPnP types time and time.tz are represented as a Long object that represents the number of ms since midnight. If the time wraps to the next day due to a time zone value, then the final value must be truncated to modulo 86.400.000.

See also *TYPE_DATE* on page 522 and further.

25.11 Configuration

In order to provide a standardized way to configure a UPnP driver bundle, the Configuration Admin property upnp.ssdp.address is defined.

The value is a String[] with a list of IP addresses, optionally followed with a colon (':', \u003A) and a port number. For example:

```
239.255.255.250:1900
```

Those addresses define the interfaces which the UPnP driver is operating on. If no SSDP address is specified, the default assumed will be 239.255.255.250:1900. If no port is specified, port 1900 is assumed as default.

25.12 Networking considerations

25.12.1 The UPnP Multicasts

The operating system must support multicasting on the selected network device. In certain cases, a multicasting route has to be set in the operating system routing table.

These configurations are highly dependent on the underlying operating system and beyond the scope of this specification.

25.13 Security

The UPnP specification is based on HTTP and uses plain text SOAP (XML) messages to control devices. For this reason, it does not provide any inherent security mechanisms. However, the UPnP specification is based on the exchange of XML files and not code. This means that at least worms and viruses cannot be implemented using the UPnP protocols.

However, a bundle registering a UPnP Device service is represented on the outside network and has the ability to communicate. The same is true for getting a UPnP Device service. It is therefore recommended that ServicePermission[REGISTER|GET,UPnPDevice|UPnPEventListener] be used sparingly and only for bundles that are trusted.

25.14 org.osgi.service.upnp

The OSGi UPnP API 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.upnp; specification-version=1.0
```

25.14.1 Summary

- *UPnPAction* – A UPnP action. [p.513]

- *UPnPDevice* – Represents a UPnP device. [p.514]
- *UPnPEventListener* – UPnP Events are mapped and delivered to applications according to the OSGi whiteboard model. [p.518]
- *UPnPIcon* – A UPnP icon representation. [p.519]
- *UPnPService* – A representation of a UPnP Service. [p.520]
- *UPnPStateVariable* – The meta-information of a UPnP state variable as declared in the device's service state table (SST). [p.522]

25.14.2 public interface UPnPAction

A UPnP action. Each UPnP service contains zero or more actions. Each action may have zero or more UPnP state variables as arguments.

25.14.2.1 public String[] getInputArgumentNames( )

□ Lists all input arguments for this action.

Each action may have zero or more input arguments.

Returns Array of input argument names or `null` if no input arguments.

See Also `UPnPStateVariable`[p.522]

25.14.2.2 public String getName( )

□ Returns the action name. The action name corresponds to the `name` field in the `actionList` of the service description.

- For standard actions defined by a UPnP Forum working committee, action names must not begin with `X_` nor `A_`.
- For non-standard actions specified by a UPnP vendor and added to a standard service, action names must begin with `X_`.

Returns Name of action, must not contain a hyphen character or a hash character

25.14.2.3 public String[] getOutputArgumentNames( )

□ List all output arguments for this action.

Returns Array of output argument names or `null` if there are no output arguments.

See Also `UPnPStateVariable`[p.522]

25.14.2.4 public String getReturnArgumentName( )

□ Returns the name of the designated return argument.

One of the output arguments can be flagged as a designated return argument.

Returns The name of the designated return argument or `null` if none is marked.

25.14.2.5 public UPnPStateVariable getStateVariable(String argumentName)

argumentName The name of the UPnP action argument.

□ Finds the state variable associated with an argument name. Helps to resolve the association of state variables with argument names in UPnP actions.

Returns State variable associated with the named argument or `null` if there is no such argument.

See Also `UPnPStateVariable`[p.522]

25.14.2.6 public Dictionary invoke(Dictionary args) throws Exception

args A Dictionary of arguments. Must contain the correct set and type of arguments for this action. May be null if no input arguments exist.

□ Invokes the action. The input and output arguments are both passed as Dictionary objects. Each entry in the Dictionary object has a String object as key representing the argument name and the value is the argument itself. The class of an argument value must be assignable from the class of the associated UPnP state variable. The input argument Dictionary object must contain exactly those arguments listed by getInputArguments method. The output argument Dictionary object will contain exactly those arguments listed by getOutputArguments method.

Returns A Dictionary with the output arguments. null if the action has no output arguments.

Throws Exception – The execution fails for some reason.

See Also UPnPStateVariable[p.522]

25.14.3 public interface UPnPDevice

Represents a UPnP device. For each UPnP root and embedded device, an object is registered with the framework under the UPnPDevice interface.

The relationship between a root device and its embedded devices can be deduced using the UPnPDevice.CHILDREN_UDN and UPnPDevice.PARENT_UDN service registration properties.

The values of the UPnP property names are defined by the UPnP Forum.

All values of the UPnP properties are obtained from the device using the device's default locale.

If an application wants to query for a set of localized property values, it has to use the method UPnPDevice.getDescriptions(String locale).

25.14.3.1 public static final String CHILDREN_UDN = "UPnP.device.childrenUDN"

The property key that must be set for all devices containing other embedded devices.

The value is an array of UDNs for each of the device's children (String[]). The array contains UDNs for the immediate descendants only.

If an embedded device in turn contains embedded devices, the latter are not included in the array.

The UPnP Specification does not encourage more than two levels of nesting.

The property is not set if the device does not contain embedded devices.

The property is of type String[]. Value is "UPnP.device.childrenUDN"

25.14.3.2 public static final String DEVICE_CATEGORY = "UPnP"

Constant for the value of the service property DEVICE_CATEGORY used for all UPnP devices. Value is "UPnP".

See Also org.osgi.service.device.Constants.DEVICE_CATEGORY

25.14.3.3 public static final String FRIENDLY_NAME = "UPnP.device.friendlyName"

Mandatory property key for a short user friendly version of the device name. The property value holds a `String` object with the user friendly name of the device. Value is "UPnP.device.friendlyName".

25.14.3.4 public static final String ID = "UPnP.device.UDN"

Property key for the Unique Device ID property. This property is an alias to `UPnPDevice.UDN`. It is merely provided for reasons of symmetry with the `UPnPService.ID` property. The value of the property is a `String` object of the Device UDN. The value of the key is "UPnP.device.UDN".

25.14.3.5 public static final String MANUFACTURER = "UPnP.device.manufacturer"

Mandatory property key for the device manufacturer's property. The property value holds a String representation of the device manufacturer's name. Value is "UPnP.device.manufacturer".

25.14.3.6 public static final String MANUFACTURER_URL = "UPnP.device.manufacturerURL"

Optional property key for a URL to the device manufacturers Web site. The value of the property is a `String` object representing the URL. Value is "UPnP.device.manufacturerURL".

25.14.3.7 public static final int MATCH_GENERIC = 1

Constant for the UPnP device match scale, indicating a generic match for the device. Value is 1.

25.14.3.8 public static final int MATCH_MANUFACTURER_MODEL = 7

Constant for the UPnP device match scale, indicating a match with the device model. Value is 7.

25.14.3.9 public static final int MATCH_MANUFACTURER_MODEL_REVISION = 15

Constant for the UPnP device match scale, indicating a match with the device revision. Value is 15.

25.14.3.10 public static final int MATCH_MANUFACTURER_MODEL_REVISION_SERIAL = 31

Constant for the UPnP device match scale, indicating a match with the device revision and the serial number. Value is 31.

25.14.3.11 public static final int MATCH_TYPE = 3

Constant for the UPnP device match scale, indicating a match with the device type. Value is 3.

25.14.3.12 public static final String MODEL_DESCRIPTION = "UPnP.device.modelDescription"

Optional (but recommended) property key for a `String` object with a long description of the device for the end user. The value is "UPnP.device.modelDescription".

25.14.3.13 **public static final String MODEL_NAME = "UPnP.device.modelName"**

Mandatory property key for the device model name. The property value holds a `String` object giving more information about the device model. Value is "UPnP.device.modelName".

25.14.3.14 **public static final String MODEL_NUMBER = "UPnP.device.modelNumber"**

Optional (but recommended) property key for a `String` class typed property holding the model number of the device. Value is "UPnP.device.modelNumber".

25.14.3.15 **public static final String MODEL_URL = "UPnP.device.modelURL"**

Optional property key for a `String` typed property holding a string representing the URL to the Web site for this model. Value is "UPnP.device.modelURL".

25.14.3.16 **public static final String PARENT_UDN = "UPnP.device.parentUDN"**

The property key that must be set for all embedded devices. It contains the UDN of the parent device. The property is not set for root devices. The value is "UPnP.device.parentUDN".

25.14.3.17 **public static final String PRESENTATION_URL = "UPnP.presentationURL"**

Optional (but recommended) property key for a `String` typed property holding a string representing the URL to a device representation Web page. Value is "UPnP.presentationURL".

25.14.3.18 **public static final String SERIAL_NUMBER = "UPnP.device.serialNumber"**

Optional (but recommended) property key for a `String` typed property holding the serial number of the device. Value is "UPnP.device.serialNumber".

25.14.3.19 **public static final String TYPE = "UPnP.device.type"**

Property key for the UPnP Device Type property. Some standard property values are defined by the Universal Plug and Play Forum. The type string also includes a version number as defined in the UPnP specification. This property must be set.

For standard devices defined by a UPnP Forum working committee, this must consist of the following components in the given order separated by colons:

- `urn`
- schemas-upnp-org
- `device`
- a device type suffix
- an integer device version

For non-standard devices specified by UPnP vendors following components must be specified in the given order separated by colons:

- `urn`
- an ICANN domain name owned by the vendor

- device
- a device type suffix
- an integer device version

To allow for backward compatibility the UPnP driver must automatically generate additional Device Type property entries for smaller versions than the current one. If for example a device announces its type as version 3, then properties for versions 2 and 1 must be automatically generated.

In the case of exporting a UPnPDevice, the highest available version must be announced on the network.

Syntax Example: urn:schemas-upnp-org:device:deviceType:v

The value is "UPnP.device.type".

25.14.3.20 public static final String UDN = "UPnP.device.UDN"

Property key for the Unique Device Name (UDN) property. It is the unique identifier of an instance of a UPnPDevice. The value of the property is a String object of the Device UDN. Value of the key is "UPnP.device.UDN". This property must be set.

25.14.3.21 public static final String UPC = "UPnP.device.UPC"

Optional property key for a String typed property holding the Universal Product Code (UPC) of the device. Value is "UPnP.device.UPC".

25.14.3.22 public static final String UPNP_EXPORT = "UPnP.export"

The UPnP.export service property is a hint that marks a device to be picked up and exported by the UPnP Service. Imported devices do not have this property set. The registered property requires no value.

The UPNP_EXPORT string is "UPnP.export".

25.14.3.23 public Dictionary getDescriptions(String locale)

locale A language tag as defined by RFC 1766 and maintained by ISO 639. Examples include "de", "en" or "en-US". The default locale of the device is specified by passing a null argument.

□ Get a set of localized UPnP properties. The UPnP specification allows a device to present different device properties based on the client's locale. The properties used to register the UPnPDevice service in the OSGi registry are based on the device's default locale. To obtain a localized set of the properties, an application can use this method.

Not all properties might be available in all locales. This method does **not** substitute missing properties with their default locale versions.

Returns Dictionary mapping property name Strings to property value Strings

25.14.3.24 public UPnPIcon[] getIcons(String locale)

locale A language tag as defined by RFC 1766 and maintained by ISO 639. Examples include "de", "en" or "en-US". The default locale of the device is specified by passing a null argument.

□ Lists all icons for this device in a given locale. The UPnP specification allows a device to present different icons based on the client's locale.

Returns Array of icons or null if no icons are available.

25.14.3.25 public UPnPService getService(String serviceId)

serviceId The service id

- Locates a specific service by its service id.

Returns The requested service or null if not found.

25.14.3.26 public UPnPService[] getServices( )

- Lists all services provided by this device.

Returns Array of services or `null` if no services are available.

25.14.4 public interface UPnPEventListener

UPnP Events are mapped and delivered to applications according to the OSGi whiteboard model. An application that wishes to be notified of events generated by a particular UPnP Device registers a service extending this interface.

The notification call from the UPnP Service to any `UPnPEventListener` object must be done asynchronous with respect to the originator (in a separate thread).

Upon registration of the UPnP Event Listener service with the Framework, the service is notified for each variable which it listens for with an initial event containing the current value of the variable. Subsequent notifications only happen on changes of the value of the variable.

A UPnP Event Listener service filter the events it receives. This event set is limited using a standard framework filter expression which is specified when the listener service is registered.

The filter is specified in a property named "upnp.filter" and has as a value an object of type `org.osgi.framework.Filter`.

When the Filter is evaluated, the folowing keywords are recognized as defined as literal constants in the `UPnPDevice` class.

The valid subset of properties for the registration of UPnP Event Listener services are:

- `UPnPDevice.TYPE` -- Which type of device to listen for events.
- `UPnPDevice.ID` -- The ID of a specific device to listen for events.
- `UPnPService.TYPE` -- The type of a specific service to listen for events.
- `UPnPService.ID` -- The ID of a specific service to listen for events.

25.14.4.1 public static final String UPNP_FILTER = "upnp.filter"

Key for a service property having a value that is an object of type `org.osgi.framework.Filter` and that is used to limit received events.

25.14.4.2 public void notifyUPnPEvent(String deviceId, String serviceId, Dictionary events)

deviceId ID of the device sending the events

serviceId ID of the service sending the events

events `Dictionary` object containing the new values for the state variables that have changed.

- Callback method that is invoked for received events. The events are collected in a `Dictionary` object. Each entry has a `String` key representing the event name (= state variable name) and the new value of the state variable. The class of the value object must match the class specified by the UPnP State Variable associated with the event. This method must be called asynchronously

25.14.5 public interface UPnPIcon

A UPnP icon representation. Each UPnP device can contain zero or more icons.

25.14.5.1 public int getDepth( )

- Returns the color depth of the icon in bits.

Returns The color depth in bits. If the actual color depth of the icon is unknown, -1 is returned.

25.14.5.2 public int getHeight( )

- Returns the height of the icon in pixels. If the actual height of the icon is unknown, -1 is returned.

Returns The height in pixels, or -1 if unknown.

25.14.5.3 public InputStream getInputStream( ) throws IOException

- Returns an `InputStream` object for the icon data. The `InputStream` object provides a way for a client to read the actual icon graphics data. The number of bytes available from this `InputStream` object can be determined via the `getSize()` method. The format of the data encoded can be determined by the MIME type availble via the `getMimeType()` method.

Returns An InputStream to read the icon graphics data from.

See Also `UPnPIcon.getMimeType()` [p.519]

25.14.5.4 public String getMimeType( )

- Returns the MIME type of the icon. This method returns the format in which the icon graphics, read from the `InputStream` object obtained by the `getInputStream()` method, is encoded.

  The format of the returned string is in accordance to RFC2046. A list of valid MIME types is maintained by the IANA at ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/media-types (ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/media-types) .

  Typical values returned include: “image/jpeg” or “image/gif”

Returns The MIME type of the encoded icon.

25.14.5.5 public int getSize( )

- Returns the size of the icon in bytes. This method returns the number of bytes of the icon available to read from the `InputStream` object obtained by the `getInputStream()` method. If the actual size can not be determined, -1 is returned.

Returns The icon size in bytes, or -1 if the size is unknown.

25.14.5.6 public int getWidth( )

- Returns the width of the icon in pixels. If the actual width of the icon is unknown, -1 is returned.

Returns The width in pixels, or -1 if unknown.

25.14.6 public interface UPnPService

A representation of a UPnP Service. Each UPnP device contains zero or more services. The UPnP description for a service defines actions, their arguments, and event characteristics.

25.14.6.1 public static final String ID = "UPnP.service.id"

Property key for the optional service id. The service id property is used when registering UPnP Device services or UPnP Event Listener services. The value of the property contains a `String` array (`String[]`) of service ids. A UPnP Device service can thus announce what service ids it contains. A UPnP Event Listener service can announce for what UPnP service ids it wants notifications. A service id does **not** have to be universally unique. It must be unique only within a device. A `null` value is a wildcard, matching **all** services. The value is "UPnP.service.id".

25.14.6.2 public static final String TYPE = "UPnP.service.type"

Property key for the optional service type uri. The service type property is used when registering UPnP Device services and UPnP Event Listener services. The property contains a `String` array (`String[]`) of service types. A UPnP Device service can thus announce what types of services it contains. A UPnP Event Listener service can announce for what type of UPnP services it wants notifications. The service version is encoded in the type string as specified in the UPnP specification. A `null` value is a wildcard, matching **all** service types. Value is "UPnP.service.type".

See Also `UPnPService.getType()` [p.521]

25.14.6.3 public UPnPAction getAction(String name)

name Name of action. Must not contain hyphen or hash characters. Should be <32 characters.

- Locates a specific action by name. Looks up an action by its name.

Returns The requested action or `null` if no action is found.

25.14.6.4 public UPnPAction[] getActions( )

- Lists all actions provided by this service.

Returns Array of actions (`UPnPAction[]`) or `null` if no actions are defined for this service.

25.14.6.5 public String getId( )

- Returns the `serviceId` field in the UPnP service description.

  For standard services defined by a UPnP Forum working committee, the serviceId must contain the following components in the indicated order:

  - `urn:upnp-org:serviceId:`
  - service ID suffix

  Example: `urn:upnp-org:serviceId:serviceID`.

  Note that `upnp-org` is used instead of `schemas-upnp-org` in this example because an XML schema is not defined for each serviceId.

  For non-standard services specified by UPnP vendors, the serviceId must contain the following components in the indicated order:

  - `urn:`
  - ICANN domain name owned by the vendor
  - `:serviceId:`
  - service ID suffix

  Example: `urn:domain-name:serviceId:serviceID`.

Returns The service ID suffix defined by a UPnP Forum working committee or specified by a UPnP vendor. Must be <= 64 characters. Single URI.

25.14.6.6 public UPnPStateVariable getStateVariable(String name)

name Name of the State Variable

- Gets a `UPnPStateVariable` objects provided by this service by name

Returns State variable or `null` if no such state variable exists for this service.

25.14.6.7 public UPnPStateVariable[] getStateVariables( )

- Lists all `UPnPStateVariable` objects provided by this service.

Returns Array of state variables or `null` if none are defined for this service.

25.14.6.8 public String getType( )

- Returns the `serviceType` field in the UPnP service description.

  For standard services defined by a UPnP Forum working committee, the serviceType must contain the following components in the indicated order:

  - `urn:schemas-upnp-org:service:`
  - service type suffix:
  - integer service version

  Example: `urn:schemas-upnp-org:service:serviceType:v`.

  For non-standard services specified by UPnP vendors, the `serviceType` must contain the following components in the indicated order:

  - `urn:`
  - ICANN domain name owned by the vendor
  - `:service:`
  - service type suffix:
  - integer service version

  Example: `urn:domain-name:service:serviceType:v`.

Returns The service type suffix defined by a UPnP Forum working committee or specified by a UPnP vendor. Must be <= 64 characters, not including the version suffix and separating colon. Single URI.

25.14.6.9 public String getVersion( )

Returns the version suffix encoded in the `serviceType` field in the UPnP service description.

Returns The integer service version defined by a UPnP Forum working committee or specified by a UPnP vendor.

25.14.7 public interface UPnPStateVariable

The meta-information of a UPnP state variable as declared in the device's service state table (SST).

Method calls to interact with a device (e.g. `UPnPAction.invoke(...);`) use this class to encapsulate meta information about the input and output arguments.

The actual values of the arguments are passed as Java objects. The mapping of types from UPnP data types to Java data types is described with the field definitions.

25.14.7.1 public static final String TYPE_BIN_BASE64 = "bin.base64"

MIME-style Base64 encoded binary BLOB.

Takes 3 Bytes, splits them into 4 parts, and maps each 6 bit piece to an octet. (3 octets are encoded as 4.) No limit on size.

Mapped to `byte[]` object. The Java byte array will hold the decoded content of the BLOB.

25.14.7.2 public static final String TYPE_BIN_HEX = "bin.hex"

Hexadecimal digits representing octets.

Treats each nibble as a hex digit and encodes as a separate Byte. (1 octet is encoded as 2.) No limit on size.

Mapped to `byte[]` object. The Java byte array will hold the decoded content of the BLOB.

25.14.7.3 public static final String TYPE_BOOLEAN = "boolean"

True or false.

Mapped to `Boolean` object.

25.14.7.4 public static final String TYPE_CHAR = "char"

Unicode string.

One character long.

Mapped to `Character` object.

25.14.7.5 public static final String TYPE_DATE = "date"

A calendar date.

Date in a subset of ISO 8601 format without time data.

See http://www.w3.org/TR/xmlschema-2/#date (http://www.w3.org/TR/xmlschema-2/#date) .

Mapped to `java.util.Date` object. Always 00:00 hours.

25.14.7.6 public static final String TYPE_DATETIME = "dateTime"

A specific instant of time.

Date in ISO 8601 format with optional time but no time zone.

See http://www.w3.org/TR/xmlschema-2/#dateTime (http://www.w3.org/TR/xmlschema-2/#dateTime) .

Mapped to `java.util.Date` object using default time zone.

25.14.7.7 public static final String TYPE_DATETIME_TZ = "dateTime.tz"

A specific instant of time.

Date in ISO 8601 format with optional time and optional time zone.

See http://www.w3.org/TR/xmlschema-2/#dateTime (http://www.w3.org/TR/xmlschema-2/#dateTime) .

Mapped to `java.util.Date` object adjusted to default time zone.

25.14.7.8 public static final String TYPE_FIXED_14_4 = "fixed.14.4"

Same as r8 but no more than 14 digits to the left of the decimal point and no more than 4 to the right.

Mapped to `Double` object.

25.14.7.9 public static final String TYPE_FLOAT = "float"

Floating-point number.

Mantissa (left of the decimal) and/or exponent may have a leading sign. Mantissa and/or exponent may have leading zeros. Decimal character in mantissa is a period, i.e., whole digits in mantissa separated from fractional digits by period. Mantissa separated from exponent by E. (No currency symbol.) (No grouping of digits in the mantissa, e.g., no commas.)

Mapped to `Float` object.

25.14.7.10 public static final String TYPE_I1 = "i1"

1 Byte int.

Mapped to `Integer` object.

25.14.7.11 public static final String TYPE_I2 = "i2"

2 Byte int.

Mapped to `Integer` object.

25.14.7.12 public static final String TYPE_I4 = "i4"

4 Byte int.

Must be between -2147483648 and 2147483647

Mapped to Integer object.

25.14.7.13 public static final String TYPE_INT = "int"

Integer number.

Mapped to Integer object.

25.14.7.14 public static final String TYPE_NUMBER = "number"

Same as r8.

Mapped to Double object.

25.14.7.15 public static final String TYPE_R4 = "r4"

4 Byte float.

Same format as float. Must be between 3.40282347E+38 to 1.17549435E-38.

Mapped to Float object.

25.14.7.16 public static final String TYPE_R8 = "r8"

8 Byte float.

Same format as float. Must be between -1.79769313486232E308 and -4.94065645841247E-324 for negative values, and between 4.94065645841247E-324 and 1.79769313486232E308 for positive values, i.e., IEEE 64-bit (8-Byte) double.

Mapped to Double object.

25.14.7.17 public static final String TYPE_STRING = "string"

Unicode string.

No limit on length.

Mapped to String object.

25.14.7.18 public static final String TYPE_TIME = "time"

An instant of time that recurs every day.

Time in a subset of ISO 8601 format with no date and no time zone.

See http://www.w3.org/TR/xmlschema-2/#time (http://www.w3.org/TR/xmlschema-2/#dateTime) .

Mapped to Long. Converted to milliseconds since midnight.

25.14.7.19 public static final String TYPE_TIME_TZ = "time.tz"

An instant of time that recurs every day.

Time in a subset of ISO 8601 format with optional time zone but no date.

See http://www.w3.org/TR/xmlschema-2/#time (http://www.w3.org/TR/xmlschema-2/#dateTime) .

Mapped to Long object. Converted to milliseconds since midnight and adjusted to default time zone, wrapping at 0 and 24*60*60*1000.

25.14.7.20 public static final String TYPE_UI1 = "ui1"

Unsigned 1 Byte int.

Mapped to an `Integer` object.

25.14.7.21 public static final String TYPE_UI2 = "ui2"

Unsigned 2 Byte int.

Mapped to `Integer` object.

25.14.7.22 public static final String TYPE_UI4 = "ui4"

Unsigned 4 Byte int.

Mapped to `Long` object.

25.14.7.23 public static final String TYPE_URI = "uri"

Universal Resource Identifier.

Mapped to `String` object.

25.14.7.24 public static final String TYPE_UUID = "uuid"

Universally Unique ID.

Hexadecimal digits representing octets. Optional embedded hyphens are ignored.

Mapped to `String` object.

25.14.7.25 public String[] getAllowedValues( )

- Returns the allowed values, if defined. Allowed values can be defined only for String types.

Returns The allowed values or `null` if not defined. Should be less than 32 characters.

25.14.7.26 public Object getDefaultValue( )

- Returns the default value, if defined.

Returns The default value or `null` if not defined. The type of the returned object can be determined by `getJavaDataType`.

25.14.7.27 public Class getJavaDataType( )

- Returns the Java class associated with the UPnP data type of this state variable.

Mapping between the UPnP data types and Java classes is performed according to the schema mentioned above.

`Integer`	`ui1, ui2, i1, i2, i4, int`
`Long`	`ui4, time, time.tz`
`Float`	`r4, float`
`Double`	`r8, number, fixed.14.4`
`Character`	`char`
`String`	`string, uri, uuid`
`Date`	`date, dateTime, dateTime.tz`

`Boolean`	`boolean`
`byte[]`	`bin.base64, bin.hex`

Returns A class object corresponding to the Java type of this argument.

25.14.7.28 public Number getMaximum( )

- Returns the maximum value, if defined. Maximum values can only be defined for numeric types.

Returns The maximum value or `null` if not defined.

25.14.7.29 public Number getMinimum( )

- Returns the minimum value, if defined. Minimum values can only be defined for numeric types.

Returns The minimum value or `null` if not defined.

25.14.7.30 public String getName( )

- Returns the variable name.
  - All standard variables defined by a UPnP Forum working committee must not begin with `X_` nor `A_`.
  - All non-standard variables specified by a UPnP vendor and added to a standard service must begin with `X_`.

Returns Name of state variable. Must not contain a hyphen character nor a hash character. Should be <32 characters.

25.14.7.31 public Number getStep( )

- Returns the size of an increment operation, if defined. Step sizes can be defined only for numeric types.

Returns The increment size or null if not defined.

25.14.7.32 public String getUPnPDataType( )

- Returns the UPnP type of this state variable. Valid types are defined as constants.

Returns The UPnP data type of this state variable, as defined in above constants.

25.14.7.33 public boolean sendsEvents( )

- Tells if this StateVariable can be used as an event source. If the StateVariable is eventable, an event listener service can be registered to be notified when changes to the variable appear.

Returns `true` if the `StateVariable` generates events, `false` otherwise.

25.15 References

[86] *UPnP Device Architecture*
http://www.upnp.org/download/UPnPDA10_20000613.htm

[87] *UPnP Forum*
http://www.upnp.org

[88] *Simple Object Access Protocol, SOAP*
http://www.w3.org/TR/SOAP

[89] *General Event Notification Architecture, GENA*
http://www.upnp.org/download/draft-cohen-gena-client-01.txt

[90] *Simple Service Discovery Protocol, SSDP*
http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt

[91] *XML Schema*
http://www.w3.org/TR/xmlschema-2

[92] *ISo 8601 Date And Time formats*
www.iso.ch

26 Initial Provisioning

Version 1.0

26.1 Introduction

To allow freedom regarding the choice of management protocol, the OSGi *Remote Management Reference Architecture* on page 29, specifies an architecture to remotely manage a Service Platform with a Management Agent. The Management Agent is implemented with a Management Bundle that can communicate with an unspecified management protocol.

This specification defines how the Management Agent can make its way to the Service Platform, and gives a structured view of the problems and their corresponding resolution methods.

The purpose of this specification is to enable the management of a Service Platform by an Operator, and (optionally) to hand over the management of the Service Platform later to another Operator. This approach is in accordance with the OSGi remote management reference architecture.

This bootstrapping process requires the installation of a Management Agent, with appropriate configuration data, in the Service Platform.

This specification consists of a prologue, in which the principles of the Initial Provisioning are outlined, and a number of mappings to different mechanisms.

26.1.1 Essentials

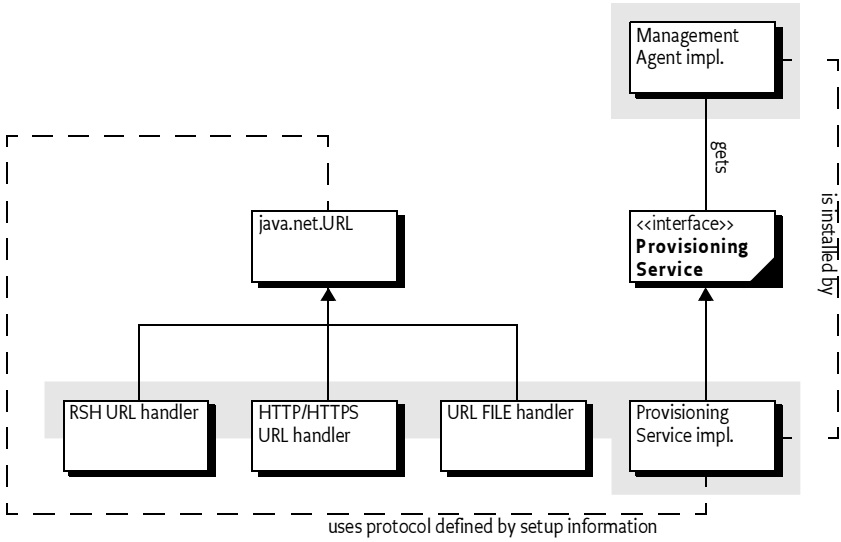
- *Policy Free* – The proposed solution must be business model agnostic; none of the affected parties (Operators, SPS Manufacturers, etc.) should be forced into any particular business model.
- *Interoperability* – The Initial Provisioning must permit arbitrary interoperability between management systems and Service Platforms. Any compliant Remote Manager should be able to manage any compliant Service Platform, even in the absence of a prior business relationship. Adhering to this requirement allows a particular Operator to manage a variety of makes and models of Service Platform Servers using a single management system of the Operator's choice. This rule also gives the consumer the greatest choice when selecting an Operator.
- *Flexible* – The management process should be as open as possible, to allow innovation and specialization while still achieving interoperability.

26.1.2 Entities

- *Provisioning Service* – A service registered with the Framework that provides information about the initial provisioning to the Management Agent.

- *Provisioning Dictionary* – A Dictionary object that is filled with information from the ZIP files that are loaded during initial setup.
- *RSH Protocol* – An OSGi specific secure protocol based on HTTP.
- *Management Agent* – A bundle that is responsible for managing a Service Platform under control of a Remote Manager.

Figure 77 *Initial Provisioning*



26.2 Procedure

The following procedure should be executed by an OSGi Framework implementation that supports this Initial Provisioning specification.

When the Service Platform is first brought under management control, it must be provided with an initial request URL in order to be provisioned. Either the end user or the manufacturer may provide the initial request URL. How the initial request URL is transferred to the Framework is not specified, but a mechanism might, for example, be a command line parameter when the framework is started.

When asked to start the Initial Provisioning, the Service Platform will send a request to the management system. This request is encoded in a URL, for example:

```
http://osgi.acme.com/remote-manager
```

This URL may use any protocol that is available on the Service Platform Server. Many standard protocols exist, but it is also possible to use a proprietary protocol. For example, software could be present which can communicate with a smart card and could handle, for example, this URL:

```
smart-card://com1:0/7F20/6F38
```

Before the request URL is executed, the Service Platform information is appended to the URL. This information includes at least the Service Platform Identifier, but may also contain proprietary information, as long as the keys for this information do not conflict. Different URL schemes may use different methods of appending parameters; these details are specified in the mappings of this specification to concrete protocols.

The result of the request must be a ZIP file (The content type should be application/zip). It is the responsibility of the underlying protocol to guarantee the integrity and authenticity of this ZIP file.

This ZIP file is unpacked and its entries (except bundle and bundle-url entries, described in Table 33) are placed in a Dictionary object. This Dictionary object is called the *Provisioning Dictionary*. It must be made available from the Provisioning Service in the service registry. The names of the entries in the ZIP file must not start with a slash ('/').

The ZIP file may contain only four types of dictionary entries: text, binary, bundle, or bundle-url. The types are specified in the ZIP entry's extra field, and must be a MIME type as defined in [99] *MIME Types*. The text and bundle-url entries are translated into a String object. All other entries must be stored as a byte[].

Type	MIME Type	Description
text	MIME_STRING text/plain;charset=utf-8	Must be represented as a String object
binary	MIME_BYTE_ARRAY application/octet-stream	Must be represented as a byte array (byte[]).
bundle	MIME_BUNDLE application/x-osgi-bundle	Entries must be installed using BundleContext.installBundle(String, InputStream), with the InputStream object constructed from the contents of the ZIP entry. The location must be the name of the ZIP entry without leading slash. This entry must not be stored in the Provisioning Dictionary.
bundle-url	MIME_BUNDLE_URL text/x-osgi-bundle-url; charset=utf-8	The content of this entry is a string coded in utf-8. Entries must be installed using BundleContext.installBundle(String, InputStream), with the InputStream object created from the given URL. The location must be the name of the ZIP entry without leading slash. This entry must not be stored in the Provisioning Dictionary.

Table 32 *Content types of provisioning ZIP file*

The Provisioning Service must install (but not start) all entries in the ZIP file that are typed in the extra field with bundle or bundle-url.

If an entry named PROVISIONING_START_BUNDLE is present in the Provisioning Dictionary, then its content type must be text as defined in Table 32. The content of this entry must match the bundle location of a previously loaded bundle. This designated bundle must be given AllPermission and started.

If no PROVISIONING_START_BUNDLE entry is present in the Provisioning Dictionary, the Provisioning Dictionary should contain a reference to another ZIP file under the PROVISIONING_REFERENCE key. If both keys are absent, no further action must take place.

If this PROVISIONING_REFERENCE key is present and holds a String object that can be mapped to a valid URL, then a new ZIP file must be retrieved from this URL. The PROVISIONING_REFERENCE link may be repeated multiple times in successively loaded ZIP files.

Referring to a new ZIP file with such a URL allows a manufacturer to place a fixed reference inside the Service Platform Server (in a file or smart card) that will provide some platform identifying information and then also immediately load the information from the management system. The PROVISIONING_REFERENCE link may be repeated multiple times in successively loaded ZIP files. The entry PROVISIONING_UPDATE_COUNT must be an Integer object that must be incremented on every iteration.

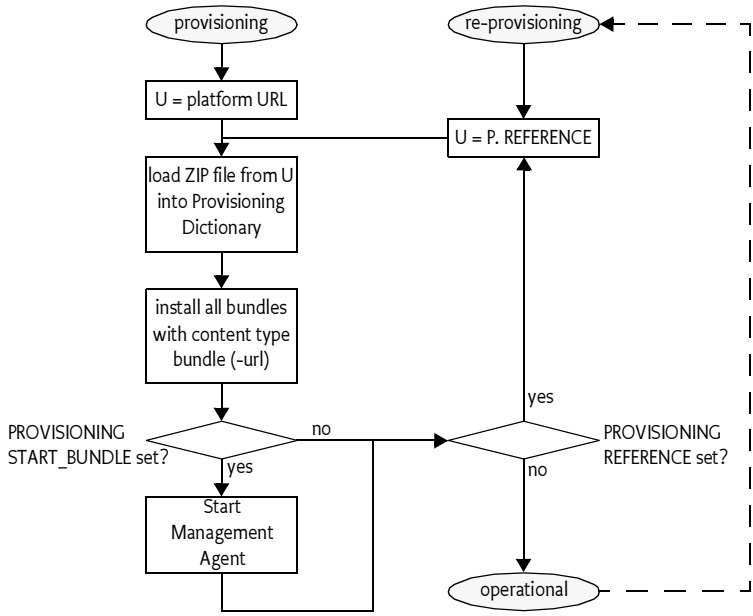
Information retrieved while loading subsequent PROVISIONING_REFERENCE URLs may replace previous key/values in the Provisioning Dictionary, but must not erase unrecognized key/values. For example, if an assignment has assigned the key proprietary-x, with a value '3', then later assignments must not override this value, unless the later loaded ZIP file contains an entry with that name. All these updates to the Provisioning Dictionary must be stored persistently. At the same time, each entry of type bundle or bundle-url (see Table 32) must be installed and not started.

Once the Management Agent has been started, the Initial Provisioning service has become operational. In this state, the Initial Provisioning service must react when the Provisioning Dictionary is updated with a new PROVISIONING_REFERENCE property. If this key is set, it should start the cycle again. For example, if the control of a Service Platform needs to be transferred to another Remote Manager, the Management Agent should set the PROVISIONING_REFERENCE to the location of this new Remote Manager's Initial Provisioning ZIP file.This process is called *re-provisioning*.

If errors occur during this process, the Initial Provisioning service should try to notify the Service User of the problem.

The previous description is depicted in Figure 78 as a flow chart.

Figure 78 *Flow chart installation Management Agent bundle*



The Management Agent may require configuration data that is specific to the Service Platform instance. If this data is available outside the Management Agent bundle, the merging of this data with the Management Agent may take place in the Service Platform. Transferring the data separately will make it possible to simplify the implementation on the server side, as it is not necessary to create *personalized* Service Platform bundles. The PROVISIONING_AGENT_CONFIG key is reserved for this purpose, but the Management Agent may use another key or mechanisms if so desired.

The PROVISIONING_SPID key must contain the Service Platform Identifier.

26.3 Special Configurations

The next section shows some examples of specially configured types of Service Platform Servers and how they are treated with the respect to the specifications in this document.

26.3.1 Branded Service Platform Server

If a Service Platform Operator is selling Service Platform Servers branded exclusively for use with their service, the provisioning will most likely be performed prior to shipping the Service Platform Server to the User. Typically the Service Platform is configured with the Dictionary entry PROVISIONING_REFERENCE pointing at a location controlled by the Operator.

Up-to-date bundles and additional configuration data must be loaded from that location at activation time. The Service Platform is probably equipped with necessary security entities, like certificates, to enable secure downloads from the Operator's URL over open networks, if necessary.

26.3.2 Non-connected Service Platform

Circumstances might exist in which the Service Platform Server has no WAN connectivity, or prefers not to depend on it for the purposes not covered by this specification.

The non-connected case can be implemented by specifying a file:// URL for the initial ZIP file (PROVISIONING_REFERENCE). That file:// URL would name a local file containing the response that would otherwise be received from a remote server.

The value for the Management Agent PROVISIONING_REFERENCE found in that file will be used as input to the load process. The PROVISIONING_REFERENCE may point to a bundle file stored either locally or remotely. No code changes are necessary for the non-connected scenario. The file:// URLs must be specified, and the appropriate files must be created on the Service Platform.

26.4 The Provisioning Service

Provisioning information is conveyed between bundles using the Provisioning Service, as defined in the ProvisioningService interface. Any changes to the Provisioning Dictionary must be propagated directly to the Provisioning Service. The Provisioning Dictionary is retrieved from the ProvisioningService object using the getInformation() method.

The Provisioning Service provides a number of methods to update the Provisioning Dictionary.

- addInformation(Dictionary) – Add all key/value pairs in the given Dictionary object to the Provisioning Dictionary.
- addInformation(ZipInputStream) – It is also possible to add a ZIP file to the Provisioning Service immediately. This will unpack the ZIP file and add the entries to the Provisioning Dictionary. This method must install the bundles contained in the ZIP file as described in *Procedure* on page 530.
- setInformation(Dictionary) – Set a new Provisioning Dictionary. This will remove all existing entries.

Each of these method will increment the PROVISIONING_UPDATE_COUNT entry.

26.5 Management Agent Environment

The Management Agent should be written with great care to minimize dependencies on other packages and services, as *all* services in OSGi are optional. Some Service Platforms may have other bundles pre-installed, so it is possible that there may be exported packages and services available. Mechanisms outside the current specification, however, must be used to discover these packages and services before the Management Agent is installed.

The Provisioning Service must ensure that the Management Agent is running with AllPermission. The Management Agent should check to see if the Permission Admin service is available, and establish the initial permissions as soon as possible to insure the security of the device when later bundles are installed. As the PermissionAdmin interfaces may not be present (it is an optional service), the Management Agent should export the PermissionAdmin interfaces to ensure they can be resolved.

Once started, the Management Agent may retrieve its configuration data from the Provisioning Service by getting the byte[] object that corresponds to the PROVISIONING_AGENT_CONFIG key in the Provisioning Dictionary. The structure of the configuration data is implementation specific.

The scope of this specification is to provide a mechanism to transmit the raw configuration data to the Management Agent. The Management Agent bundle may alternatively be packaged with its configuration data in the bundle, so it may not be necessary for the Management Agent bundle to use the Provisioning Service at all.

Most likely, the Management Agent bundle will install other bundles to provision the Service Platform. Installing other bundles might even involve downloading a more full featured Management Agent to replace the initial Management Agent.

26.6 Mapping To File Scheme

The file: scheme is the simplest and most completely supported scheme which can be used by the Initial Provisioning specification. It can be used to store the configuration data and Management Agent bundle on the Service Platform Server, and avoids any outside communication.

If the initial request URL has a file scheme, no parameters should be appended, because the file: scheme does not accept parameters.

26.6.1 Example With File Scheme

The manufacturer should prepare a ZIP file containing only one entry named PROVISIONING_START_BUNDLE that contains a location string of an entry of type application/x-osgi-bundle or application/x-osgi-bundle-URL. For example, the following ZIP file demonstrates this:

```
provisioning.start.bundle  text      agent
agent                      bundle    CoAFoE9B2AB..
```

The bundle may also be specified with a URL:

```
provisioning.start.bundle  text       http://acme.com/a.jar
agent                      bundle-url http://acme.com/a.jar
```

Upon startup, the framework is provided with the URL with the file: scheme that points to this ZIP file:

```
file:/opt/osgi/ma.zip
```

26.7 Mapping To HTTP(S) Scheme

This section defines how HTTP and HTTPS URLs must be used with the Initial Provisioning specification.

- HTTP – May be used when the data exchange takes place over networks that are secured by other means, such as a Virtual Private Network (VPN) or a physically isolated network. Otherwise, HTTP is not a valid scheme because no authentication takes place.
- HTTPS – May be used if the Service Platform is equipped with appropriate certificates.

HTTP and HTTPS share the following qualities:

- Both are well known and widely used
- Numerous implementations of the protocols exist
- Caching of the Management Agent will be desired in many implementations where limited bandwidth is an issue. Both HTTP and HTTPS already contain an accepted protocol for caching.

Both HTTP and HTTPS must be used with the GET method. The response is a ZIP file, implying that the response header `Content-Type` header must contain `application/zip`.

26.7.1 HTTPS Certificates

In order to use HTTPS, certificates must be in place. These certificates, that are used to establish trust towards the Operator, may be made available to the Service Platform using the Provisioning Service. The root certificate should be assigned to the Provisioning Dictionary before the HTTPS provider is used. Additionally, the Service Platform should be equipped with a Service Platform certificate that allows the Service Platform to properly authenticate itself towards the Operator. This specification does not state how this certificate gets installed into the Service Platform.

The root certificate is stored in the Provisioning Dictionary under the key:

```
PROVISIONING_ROOTX509
```

The Root X.509 Certificate holds certificates used to represent a handle to a common base for establishing trust. The certificates are typically used when authenticating a Remote Manager to the Service Platform. In this case, a Root X.509 certificate must be part of a certificate chain for the Operator's certificate. The format of the certificate is defined in *Certificate Encoding* on page 537.

26.7.2 Certificate Encoding

Root certificates are X.509 certificates. Each individual certificate is stored as a byte[] object. This byte[] object is encoded in the default Java manner, as follows:

- The original, binary certificate data is DER encoded
- The DER encoded data is encoded into base64 to make it text.
- The base64 encoded data is prefixed with
  -----BEGIN CERTIFICATE-----
  and suffixed with:
  -----END CERTIFICATE-----
- If a record contains more than one certificate, they are simply appended one after the other, each with a delimiting prefix and suffix.

The decoding of such a certificate may be done with the java.security.cert.CertificateFactory class:

```
InputStream bis = new ByteArrayInputStream(x509); // byte[]
CertificateFactory cf =
   CertificateFactory.getInstance("X.509");
Collection c = cf.generateCertificates(bis);
Iterator i = c.iterator();
while (i.hasNext()) {
   Certificate cert = (Certificate)i.next();
   System.out.println(cert);
}
```

26.7.3 URL Encoding

The URL must contain the Service Platform Identity, and may contain more parameters. These parameters are encoded in the URL according to the HTTP(S) URL scheme. A base URL may be set by an end user but the Provisioning Service must add the Service Platform Identifier.

If the request URL already contains HTTP parameters (if there is a '?' in the request), the service_platform_id is appended to this URL as an additional parameter. If, on the other hand, the request URL does not contain any HTTP parameters, the service_platform_id will be appended to the URL after a '?', becoming the first HTTP parameter. The following two examples show these two variants:

```
http://server.operator.com/service-x? «
   foo=bar&service_platform_id=VIN:123456789

http://server.operator.com/service-x? «
   service_platform_id=VIN:123456789
```

Proper URL encoding must be applied when the URL contains characters that are not allowed. See [98] *RFC 2396 - Uniform Resource Identifier (URI).*

26.8 Mapping To RSH Scheme

The RSH protocol is an OSGi-specific protocol, and is included in this specification because it is optimized for Initial Provisioning. It requires a shared secret between the management system and the Service Platform that is small enough to be entered by the Service User.

RSH bases authentication and encryption on Message Authentication Codes (MACs) that have been derived from a secret that is shared between the Service Platform and the Operator prior to the start of the protocol execution.

The protocol is based on an ordinary HTTP GET request/response, in which the request must be *signed* and the response must be *encrypted* and *authenticated*. Both the *signature* and *encryption key* are derived from the shared secret using Hashed Message Access Codes (HMAC) functions.

As additional input to the HMAC calculations, one client-generated nonce and one server-generated nonce are used to prevent replay attacks. The nonces are fairly large random numbers that must be generated in relation to each invocation of the protocol, in order to guarantee freshness. These nonces are called `clientfg` (client-generated freshness guarantee) and `serverfg` (server-generated freshness guarantee).

In order to separate the HMAC calculations for authentication and encryption, each is based on a different constant value. These constants are called the *authentication constant* and the *encryption constant*.

From an abstract perspective, the protocol may be described as follows.

- δ – Shared secret, 160 bits or more
- *s* – Server nonce, called `servercfg`, 128 bits
- *c* – Client nonce, called `clientfg`, 128 bits
- K_a – Authentication key, 160 bits
- K_e – Encryption key, 192 bits
- *r* – Response data
- *e* – Encrypted data
- *E* – Encryption constant, a `byte[]` of 05, 36, 54, 70, 00 (hex)
- *A* – Authentication constant, a `byte[]` of 00, 4f, 53, 47, 49 (hex)
- *M* – Message material, used for K_e calculation.
- *m* – The calculated message authentication code.
- *3DES* – Triple DES, encryption function, see [100] *3DES*. The bytes of the key must be set to odd parity. CBC mode must be used where the padding method is defined in [101] *RFC 1423 Part III: Algorithms, Modes, and Identifiers*. In [103] *Java Cryptography API (part of Java 1.4)* this is addressed as `PKCS5Padding`.
- *IV* – Initialization vector for 3DES.
- *SHA1* – Secure Hash Algorithm to generate the Hashed Message Autentication Code, see [104] *SHA-1*. The function takes a single parameter, the block to be worked upon.
- *HMAC* – The fuction that calculates a message authentication code, which must HMAC-SHA1. HMAC-SHA1 is defined in [93] *HMAC: Keyed-Hashing for Message Authentication*. The HMAC function takes a key and a block to be worked upon as arguments. Note that the lower 16 bytes of the result must be used.

- *{}* – Concatenates its arguments
- *[]* – Indicates access to a sub-part of a variable, in bytes. Index starts at one, not zero.

In each step, the emphasized server or client indicates the context of the calculation. If both are used at the same time, each variable will have server or client as a subscript.

1. The *client* generates a random nonce, stores it and denotes it clientfg

   $c = nonce$

2. The client sends the request with the clientfg to the server.

   $c_{server} \Leftarrow c_{client}$

3. The *server* generates a nonce and denotes it serverfg.

   $s = nonce$

4. The *server* calculates an authentication key based on the SHA1 function, the shared secret, the received clientfg, the serverfg and the authentication constant.

   $K_a \leftarrow SHA1(\{\delta, c, s, A\})$

5. The *server* calculates an encryption key using an SHA-1 function, the shared secret, the received clientfg, the serverfg and the encryption constant. It must first calculate the *key material* M.

   $M[1, 20] \leftarrow SHA1(\{\delta, c, s, E\})$

   $M[21, 40] \leftarrow SHA1(\{\delta, M[1, 20], c, s, E\})$

6. The key for DES consists K_e and IV.

   $K_e \leftarrow M[1, 24]$

   $IV \leftarrow M[25, 32]$

   The *server* encrypts the response data using the encryption key derived in 5. The encryption algorithm that must be used to encrypt/decrypt the response data is 3DES. 24 bytes (192 bits) from M are used to generate K_e, but the low order bit of each byte must be used as an odd parity bit. This means that before using K_e, each byte must be processed to set the low order bit so that the byte has odd parity.

   The encryption/decryption key used is specified by the following:

   $e \leftarrow 3DES(K_e, IV, r)$

7. The *server* calculates a MAC *m* using the HMAC function, the encrypted response data and the authentication key derived in 4.

   $m \leftarrow HMAC(K_a, e)$

8. The *server* sends a response to the *client* containing the serverfg, the MAC *m* and the encrypted response data

   $s_{client} \Leftarrow s_{server}$

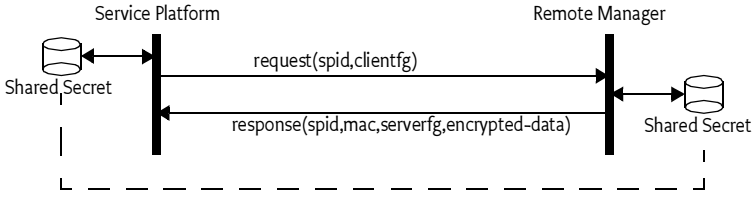
   $m_{client} \Leftarrow m_{server}$

   $e_{client} \Leftarrow e_{server}$

The *client* calculates the encryption key K_e the same way the server did in step 5 and 6. and uses this to decrypt the encrypted response data. The serverfg value received in the response is used in the calculation.
$r \leftarrow 3DES(K_e, IV, e)$

9. The *client* performs the calculation of the MAC *m'* in the same way the server did, and checks that the results match the received MAC *m*. If they do not match, further processing is discarded. The serverfg value received in the response is used in the calculation.
$K_a \leftarrow SHA1(\{\delta, c, s, A\})$
$m' \leftarrow HMAC(K_a, e)$
$m' = m$

Figure 79 *Action Diagram for RSH*



26.8.1 Shared Secret

The *shared secret* should be a key of length 160 bits (20 bytes) or more. The length is selected to match the output of the selected hash algorithm [94] *NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995.*.

In some scenarios, the shared secret is generated by the Operator and communicated to the User, who inserts the secret into the Service Platform through some unspecified means.

The opposite is also possible: the shared secret can be stored within the Service Platform, extracted from it, and then communicated to the Operator. In this scenario, the source of the shared secret could be either the Service Platform or the Operator.

In order for the server to calculate the authentication and encryption keys, it requires the proper shared secret. The server must have access to many different shared secrets, one for each Service Platform it is to support. To be able to resolve this issue, the server must typically also have access to the Service Platform Identifier of the Service Platform. The normal way for the server to know the Service Platform Identifier is through the application protocol, as this value is part of the URL encoded parameters of the HTTP, HTTPS, or RSH mapping of the Initial Provisioning.

In order to be able to switch Operators, a new shared secret must be used. The new secret may be generated by the new Operator and then inserted into the Service Platform device using a mechanism not covered by this specification. Or the device itself may generate the new secret and convey it

to the owner of the device using a display device or read-out, which is then communicated to the new operator out-of-band. Additionally, the generation of the new secret may be triggered by some external event, like holding down a button for a specified amount of time.

26.8.2 Request Coding

RSH is mapped to HTTP or HTTPS. Thus, the request parameters are URL encoded as discussed in 26.7.3 *URL Encoding*. RSH requires an additional parameter in the URL: the clientfg parameter. This parameter is a nonce that is used to counter replay attacks. See also *RSH Transport* on page 542.

26.8.3 Response Coding

The server's response to the client is composed of three parts:

- A header containing the protocol version and the serverfg
- The MAC
- The encrypted response

These three items are packaged into a binary container according to Table 33.

Bytes	Description	Value hex
4	Number of bytes in header	2E
2	Version	01 00
16	serverfg	...
4	Number of bytes in MAC	10
16	Message Authentication Code	MAC
4	Number of bytes of encrypted ZIP file	N
N	Encrypted ZIP file	...

Table 33 *RSH Header description*

The response content type is an RSH-specific encrypted ZIP file, implying that the response header Content-Type must be application/x-rsh for the HTTP request. When the content file is decrypted, the content must be a ZIP file.

26.8.4 RSH URL

The RSH URL must be used internally within the Service Platform to indicate the usage of RSH for initial provisioning. The RSH URL format is identical to the HTTP URL format, except that the scheme is rsh: instead of http:. For example (« means line continues on next line):

```
rsh://server.operator.com/service-x
```

26.8.5 Extensions to the Provisioning Service Dictionary

RSH specifies one additional entry for the Provisioning Dictionary:

PROVISIONING_RSH_SECRET

The value of this entry is a byte[] containing the shared secret used by the RSH protocol.

26.8.6 RSH Transport

RSH is mapped to HTTP or HTTPS and follows the same URL encoding rules, except that the clientfg is additionally appended to the URL. The key in the URL must be clientfg and the value must be encoded in base 64 format:

The clientfg parameter is transported as an HTTP parameter that is appended after the service_platform_id parameter. The second example above would then be:

```
rsh://server.operator.com/service-x
```

Which, when mapped to HTTP, must become:

```
http://server.operator.com/service-x? «
   service_platform_id=VIN:123456789& «
   clientfg=AHPmWcw%2FsiWYC37xZNdKvQ%3D%3D
```

26.9 Security

The security model for the Service Platform is based on the integrity of the Management Agent deployment. If any of the mechanisms used during the deployment of management agents are weak, or can be compromised, the whole security model becomes weak.

From a security perspective, one attractive means of information exchange would be a smart card. This approach enables all relevant information to be stored in a single place. The Operator could then provide the information to the Service Platform by inserting the smart card into the Service Platform.

26.9.1 Concerns

The major security concerns related to the deployment of the Management Agent are:

- The Service Platform is controlled by the intended Operator
- The Operator controls the intended Service Platform(s)
- The integrity and confidentiality of the information exchange that takes place during these processes must be considered

In order to address these concerns, an implementation of the OSGi Remote Management Architecture must assure that:

- The Operator authenticates itself to the Service Platform
- The Service Platform authenticates itself to the Operator
- The integrity and confidentiality of the Management Agent, certificates, and configuration data are fully protected if they are transported over public transports.

Each mapping of the Initial Provisioning specification to a concrete implementation must describe how these goals are met.

26.9.2 Service Platform Long-Term Security

Secrets for long-term use may be exchanged during the Initial Provisioning procedures. This way, one or more secrets may be shared securely, assuming that the Provisioning Dictionary assignments used are implemented with the proper security characteristics.

26.9.3 Permissions

The provisioning information may contain sensitive information. Also, the ability to modify provisioning information can have drastic consequences. Thus, only trusted bundles should be allowed to register, or get the Provisioning Service. This restriction can be enforced using `ServicePermission[GET, ProvisioningService]`.

No `Permission` classes guard reading or modification of the Provisioning Dictionary, so care must be taken not to leak the `Dictionary` object received from the Provisioning Service to bundles that are not trusted.

Whether message-based or connection-based, the communications used for Initial Provisioning must support mutual authentication and message integrity checking, at a minimum.

By using both server and client authentication in HTTPS, the problem of establishing identity is solved. In addition, HTTPS will encrypt the transmitted data. HTTPS requires a Public Key Infrastructure implementation in order to retrieve the required certificates.

When RSH is used, it is vital that the shared secret is shared only between the Operator and the Service Platform, and no one else.

26.10 org.osgi.service.provisioning

The OSGi Provisioning Service 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.provisioning; specification-version=1.0
```

26.10.1 public interface ProvisioningService

Service for managing the initial provisioning information.

Initial provisioning of an OSGi device is a multi step process that culminates with the installation and execution of the initial management agent. At each step of the process, information is collected for the next step. Multiple bundles may be involved and this service provides a means for these bundles to exchange information. It also provides a means for the initial Management Bundle to get its initial configuration information.

The provisioning information is collected in a `Dictionary` object, called the Provisioning Dictionary. Any bundle that can access the service can get a reference to this object and read and update provisioning information. The key of the dictionary is a `String` object and the value is a `String` or `byte[]`

object. The single exception is the PROVISIONING_UPDATE_COUNT value which is an Integer. The `provisioning` prefix is reserved for keys defined by OSGi, other key names may be used for implementation dependent provisioning systems.

Any changes to the provisioning information will be reflected immediately in all the dictionary objects obtained from the Provisioning Service.

Because of the specific application of the Provisioning Service, there should be only one Provisioning Service registered. This restriction will not be enforced by the Framework. Gateway operators or manufactures should ensure that a Provisioning Service bundle is not installed on a device that already has a bundle providing the Provisioning Service.

The provisioning information has the potential to contain sensitive information. Also, the ability to modify provisioning information can have drastic consequences. Thus, only trusted bundles should be allowed to register and get the Provisioning Service. The `ServicePermission` is used to limit the bundles that can gain access to the Provisioning Service. There is no check of `Permission` objects to read or modify the provisioning information, so care must be taken not to leak the Provisioning Dictionary received from `getInformation` method.

26.10.1.1 public static final String MIME_BUNDLE = "application/x-osgi-bundle"

MIME type to be stored in the extra field of a `ZipEntry` object for an installable bundle file. Zip entries of this type will be installed in the framework, but not started. The entry will also not be put into the information dictionary.

26.10.1.2 public static final String MIME_BUNDLE_URL = "text/x-osgi-bundle-url"

MIME type to be stored in the extra field of a ZipEntry for a String that represents a URL for a bundle. Zip entries of this type will be used to install (but not start) a bundle from the URL. The entry will not be put into the information dictionary.

26.10.1.3 public static final String MIME_BYTE_ARRAY = "application/octet-stream"

MIME type to be stored in the extra field of a `ZipEntry` object for `byte[]` data.

26.10.1.4 public static final String MIME_STRING = "text/plain;charset=utf-8"

MIME type to be stored in the extra field of a `ZipEntry` object for String data.

26.10.1.5 public static final String PROVISIONING_AGENT_CONFIG = "provisioning.agent.config"

The key to the provisioning information that contains the initial configuration information of the initial Management Agent. The value will be of type `byte[]`.

26.10.1.6 public static final String PROVISIONING_REFERENCE =

"provisioning.reference"

The key to the provisioning information that contains the location of the provision data provider. The value must be of type `String`.

26.10.1.7 public static final String PROVISIONING_ROOTX509 = "provisioning.rootx509"

The key to the provisioning information that contains the root X509 certificate used to esatblish trust with operator when using HTTPS.

26.10.1.8 public static final String PROVISIONING_RSH_SECRET = "provisioning.rsh.secret"

The key to the provisioning information that contains the shared secret used in conjunction with the RSH protocol.

26.10.1.9 public static final String PROVISIONING_SPID = "provisioning.spid"

The key to the provisioning information that uniquely identifies the Service Platform. The value must be of type `String`.

26.10.1.10 public static final String PROVISIONING_START_BUNDLE = "provisioning.start.bundle"

The key to the provisioning information that contains the location of the bundle to start with `AllPermission`. The bundle must have be previously installed for this entry to have any effect.

26.10.1.11 public static final String PROVISIONING_UPDATE_COUNT = "provisioning.update.count"

The key to the provisioning information that contains the update count of the info data. Each set of changes to the provisioning information must end with this value being incremented. The value must be of type `Integer`. This key/value pair is also reflected in the properties of the ProvisioningService in the service registry.

26.10.1.12 public void addInformation(Dictionary info)

info the set of Provisioning Information key/value pairs to add to the Provisioning Information dictionary. Any keys are values that are of an invalid type will be silently ignored.

- Adds the key/value pairs contained in `info` to the Provisioning Information dictionary. This method causes the `PROVISIONING_UPDATE_COUNT` to be incremented.

26.10.1.13 public void addInformation(ZipInputStream zis) throws IOException

zis the `ZipInputStream` that will be used to add key/value pairs to the Provisioning Information dictionary and install and start bundles. If a `ZipEntry` does not have an `Extra` field that corresponds to one of the four defined MIME types (`MIME_STRING`, `MIME_BYTE_ARRAY`, `MIME_BUNDLE`, and `MIME_BUNDLE_URL`) in will be silently ignored.

- Processes the ZipInputStream and extracts information to add to the Provisioning Information dictionary, as well as, install/update and start bundles. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

Throws IOException – if an error occurs while processing the ZipInputStream. No additions will be made to the Provisioning Information dictionary and no bundles must be started or installed.

26.10.1.14 public Dictionary getInformation( )

- Returns a reference to the Provisioning Dictionary. Any change operations (put and remove) to the dictionary will cause an UnsupportedOperationException to be thrown. Changes must be done using the setInformation and addInformation methods of this service.

26.10.1.15 public void setInformation(Dictionary info)

info the new set of Provisioning Information key/value pairs. Any keys are values that are of an invalid type will be silently ignored.

- Replaces the Provisioning Information dictionary with the key/value pairs contained in info. Any key/value pairs not in info will be removed from the Provisioning Information dictionary. This method causes the PROVISIONING_UPDATE_COUNT to be incremented.

26.11 References

[93] *HMAC:* Keyed-Hashing for Message Authentication http://www.ietf.org/rfc/rfc2104.txt Krawczyk ,et. al. 1997.

[94] *NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995.*

[95] *Hypertext Transfer Protocol - HTTP/1.1* http://www.ietf.org/rfc/rfc2616.txt *Fielding, R., et. al.*

[96] *Rescorla, E., HTTP over TLS, IETF RFC 2818, May 2000* http://www.ietf.org/rfc/rfc2818.txt.

[97] *ZIP Archive format* ftp://ftp.uu.net/pub/archiving/zip/doc/appnote-970311-iz.zip

[98] *RFC 2396 - Uniform Resource Identifier (URI)* http://www.ietf.org/rfc/rfc2396.txt

[99] *MIME Types* http://www.ietf.org/rfc/rfc2046.txt and http://www.iana.org/assignments/media-types

[100] *3DES* W/ Tuchman, "Hellman Presents No Shortcut Solution to DES," IEEE Spectrum, v. 16, n. 7 July 1979, pp40-41.

[101] *RFC 1423 Part III: Algorithms, Modes, and Identifiers* http://www.ietf.org/rfc/rfc1423.txt

[102] *PKCS 5* ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2

[103] *Java Cryptography API (part of Java 1.4)* http://java.sun.com/products/jce/index-14.html

[104] *SHA-1*
U.S. Government, Proposed Federal Information Processing Standard for Secure Hash Standard, January 1992

[105] *Transport Layer Security*
http://www.ietf.org/rfc/rfc2246.txt, January 1999, The TLS Protocol Version 1.0, T. Dierks & C. Allen.

27 Method Overview

This chapter contains a list of all the packages, classes, and methods of specifications that are part of the OSGi Service Platform, Release 3.

27.0.1 Method Index org.osgi.framework

final class AdminPermission extends BasicPermission

AdminPermission()
AdminPermission(String,String)
boolean equals(Object)
boolean implies(Permission)
PermissionCollection newPermissionCollection()

interface Bundle

static final int ACTIVE
static final int INSTALLED
static final int RESOLVED
static final int STARTING
static final int STOPPING
static final int UNINSTALLED
long getBundleId()
Dictionary getHeaders()
String getLocation()
ServiceReference[] getRegisteredServices()
URL getResource(String)
ServiceReference[] getServicesInUse()
int getState()
boolean hasPermission(Object)
void start() throws BundleException
void stop() throws BundleException
void uninstall() throws BundleException
void update() throws BundleException
void update(InputStream) throws BundleException

interface BundleActivator

void start(BundleContext) throws Exception
void stop(BundleContext) throws Exception

interface BundleContext

void addBundleListener(BundleListener)
void addFrameworkListener(FrameworkListener)
void addServiceListener(ServiceListener, String) throws InvalidSyntaxException
void addServiceListener(ServiceListener)
Filter createFilter(String) throws InvalidSyntaxException
Bundle getBundle()
Bundle getBundle(long)
Bundle[] getBundles()
File getDataFile(String)
String getProperty(String)
Object getService(ServiceReference)
ServiceReference getServiceReference(String)
ServiceReference[] getServiceReferences(String,String) throws InvalidSyntaxException
Bundle installBundle(String) throws BundleException
Bundle installBundle(String,InputStream) throws BundleException
ServiceRegistration registerService(String[], Object,Dictionary)
ServiceRegistration registerService(String, Object,Dictionary)
void removeBundleListener(BundleListener)
void removeFrameworkListener(FrameworkListener)
void removeServiceListener(ServiceListener)
boolean ungetService(ServiceReference)

class BundleEvent extends EventObject

static final int INSTALLED
static final int STARTED
static final int STOPPED
static final int UNINSTALLED
static final int UPDATED
BundleEvent(int,Bundle)
Bundle getBundle()
int getType()

class BundleException extends Exception

BundleException(String,Throwable)
BundleException(String)
Throwable getNestedException()

interface BundleListener extends EventListener

void bundleChanged(BundleEvent)

interface Configurable

Object getConfigurationObject()

interface Constants

static final String BUNDLE_ACTIVATOR
static final String BUNDLE_CATEGORY
static final String BUNDLE_CLASSPATH
static final String BUNDLE_CONTACTADDRESS
static final String BUNDLE_COPYRIGHT
static final String BUNDLE_DESCRIPTION
static final String BUNDLE_DOCURL
static final String BUNDLE_NAME
static final String BUNDLE_NATIVECODE
static final String BUNDLE_NATIVECODE_LANGUAGE
static final String BUNDLE_NATIVECODE_OSNAME
static final String BUNDLE_NATIVECODE_OSVERSION
static final String BUNDLE_NATIVECODE_PROCESSOR

static final String BUNDLE_REQUIREDEXECUTIONENVIRONMENT
static final String BUNDLE_UPDATELOCATION
static final String BUNDLE_VENDOR
static final String BUNDLE_VERSION
static final String DYNAMICIMPORT_PACKAGE
static final String EXPORT_PACKAGE
static final String EXPORT_SERVICE
static final String FRAMEWORK_EXECUTIONENVIRONMENT
static final String FRAMEWORK_LANGUAGE
static final String FRAMEWORK_OS_NAME
static final String FRAMEWORK_OS_VERSION
static final String FRAMEWORK_PROCESSOR
static final String FRAMEWORK_VENDOR
static final String FRAMEWORK_VERSION
static final String IMPORT_PACKAGE
static final String IMPORT_SERVICE
static final String OBJECTCLASS
static final String PACKAGE_SPECIFICATION_VERSION
static final String SERVICE_DESCRIPTION
static final String SERVICE_ID
static final String SERVICE_PID
static final String SERVICE_RANKING
static final String SERVICE_VENDOR
static final String SYSTEM_BUNDLE_LOCATION

interface Filter
boolean equals(Object)
int hashCode()
boolean match(ServiceReference)
boolean match(Dictionary)
String toString()

class FrameworkEvent extends EventObject
static final int ERROR
static final int PACKAGES_REFRESHED
static final int STARTED
static final int STARTLEVEL_CHANGED
FrameworkEvent(int,Object)
FrameworkEvent(int,Bundle,Throwable)
Bundle getBundle()
Throwable getThrowable()
int getType()

interface FrameworkListener extends EventListener
void frameworkEvent(FrameworkEvent)

class InvalidSyntaxException extends Exception
InvalidSyntaxException(String,String)
String getFilter()

final class PackagePermission extends BasicPermission
static final String EXPORT
static final String IMPORT
PackagePermission(String,String)
boolean equals(Object)
String getActions()
int hashCode()
boolean implies(Permission)
PermissionCollection newPermissionCollection()

class ServiceEvent extends EventObject
static final int MODIFIED
static final int REGISTERED
static final int UNREGISTERING
ServiceEvent(int,ServiceReference)
ServiceReference getServiceReference()
int getType()

interface ServiceFactory
Object getService(Bundle, ServiceRegistration)
void ungetService(Bundle, ServiceRegistration,Object)

interface ServiceListener extends EventListener
void serviceChanged(ServiceEvent)

final class ServicePermission extends BasicPermission
static final String GET
static final String REGISTER
ServicePermission(String,String)
boolean equals(Object)
String getActions()
int hashCode()
boolean implies(Permission)
PermissionCollection newPermissionCollection()

interface ServiceReference
Bundle getBundle()
Object getProperty(String)
String[] getPropertyKeys()
Bundle[] getUsingBundles()

interface ServiceRegistration
ServiceReference getReference()
void setProperties(Dictionary)
void unregister()

interface SynchronousBundleListener extends BundleListener

27.0.2 org.osgi.service.cm

interface Configuration
void delete() throws IOException
boolean equals(Object)
String getBundleLocation()
String getFactoryPid()
String getPid()
Dictionary getProperties()
int hashCode()
void setBundleLocation(String)
void update(Dictionary) throws IOException
void update() throws IOException

interface ConfigurationAdmin
static final String SERVICE_BUNDLELOCATION
static final String SERVICE_FACTORYPID

Configuration
createFactoryConfiguration(String)
throws IOException
Configuration
createFactoryConfiguration(String,
String) throws IOException
Configuration getConfiguration(String,
String) throws IOException
Configuration getConfiguration(String)
throws IOException
Configuration[] listConfigurations(String)
throws IOException,
InvalidSyntaxException

class ConfigurationException extends Exception
ConfigurationException(String,String)
String getProperty()
String getReason()

interface ConfigurationPlugin
static final String CM_TARGET
void modifyConfiguration(ServiceReference,
Dictionary)

interface ManagedService
void updated(Dictionary) throws
ConfigurationException

interface ManagedServiceFactory
void deleted(String)
String getName()
void updated(String,Dictionary) throws
ConfigurationException

27.0.3 org.osgi.service.device

interface Constants
static final String DEVICE_CATEGORY
static final String DEVICE_DESCRIPTION
static final String DEVICE_SERIAL
static final String DRIVER_ID

interface Device
static final int MATCH_NONE
void noDriverFound()

interface Driver
String attach(ServiceReference) throws
Exception
int match(ServiceReference) throws
Exception

interface DriverLocator
String[] findDrivers(Dictionary)
InputStream loadDriver(String) throws
IOException

interface DriverSelector
static final int SELECT_NONE
int select(ServiceReference,Match[])

interface Match
ServiceReference getDriver()
int getMatchValue()

27.0.4 org.osgi.service.http

interface HttpContext
static final String AUTHENTICATION_TYPE
static final String AUTHORIZATION
static final String REMOTE_USER
String getMimeType(String)
URL getResource(String)
boolean handleSecurity(HttpServletRequest,
HttpServletResponse) throws
IOException

interface HttpService
HttpContext createDefaultHttpContext()
void registerResources(String,String,
HttpContext) throws
NamespaceException
void registerServlet(String,Servlet,
Dictionary,HttpContext) throws
ServletException, NamespaceException
void unregister(String)

class NamespaceException extends Exception
NamespaceException(String)
NamespaceException(String,Throwable)
Throwable getException()

27.0.5 org.osgi.service.io

interface ConnectionFactory
static final String IO_SCHEME
Connection createConnection(String,int,
boolean) throws IOException

interface ConnectorService
static final int READ
static final int READ_WRITE
static final int WRITE
Connection open(String) throws IOException
Connection open(String,int) throws
IOException
Connection open(String,int,boolean) throws
IOException
DataInputStream
openDataInputStream(String) throws
IOException
DataOutputStream
openDataOutputStream(String) throws
IOException
InputStream openInputStream(String) throws
IOException
OutputStream openOutputStream(String)
throws IOException

27.0.6 org.osgi.service.jini

interface JiniDriver

static final String CM_LUS_EXPORT_GROUPS
static final String CM_LUS_IMPORT_GROUPS
static final String DEVICE_CATEGORY
static final String ENTRIES
static final String EXPORT
static final String LUS_EXPORT_GROUPS
static final String SERVICE_ID
ServiceTemplate[] getServiceTemplates()
void setServiceTemplates(ServiceTemplate[])

27.0.7 org.osgi.service.log

interface LogEntry

Bundle getBundle()
Throwable getException()
int getLevel()
String getMessage()
ServiceReference getServiceReference()
long getTime()

interface LogListener extends EventListener

void logged(LogEntry)

interface LogReaderService

void addLogListener(LogListener)
Enumeration getLog()
void removeLogListener(LogListener)

interface LogService

static final int LOG_DEBUG
static final int LOG_ERROR
static final int LOG_INFO
static final int LOG_WARNING
void log(int,String)
void log(int,String,Throwable)
void log(ServiceReference,int,String)
void log(ServiceReference,int,String, Throwable)

27.0.8 org.osgi.service.metatype

interface AttributeDefinition

static final int BIGDECIMAL
static final int BIGINTEGER
static final int BOOLEAN
static final int BYTE
static final int CHARACTER
static final int DOUBLE
static final int FLOAT
static final int INTEGER
static final int LONG
static final int SHORT
static final int STRING
int getCardinality()
String[] getDefaultValue()
String getDescription()
String getID()
String getName()
String[] getOptionLabels()
String[] getOptionValues()
int getType()
String validate(String)

interface MetaTypeProvider

String[] getLocales()
ObjectClassDefinition getObjectClassDefinition(String,String)

interface ObjectClassDefinition

static final int ALL
static final int OPTIONAL
static final int REQUIRED
AttributeDefinition[] getAttributeDefinitions(int)
String getDescription()
InputStream getIcon(int) throws IOException
String getID()
String getName()

27.0.9 org.osgi.service.packageadmin

interface ExportedPackage

Bundle getExportingBundle()
Bundle[] getImportingBundles()
String getName()
String getSpecificationVersion()
boolean isRemovalPending()

interface PackageAdmin

ExportedPackage getExportedPackage(String)
ExportedPackage[] getExportedPackages(Bundle)
void refreshPackages(Bundle[])

27.0.10 org.osgi.service.permissionadmin

interface PermissionAdmin

PermissionInfo[] getDefaultPermissions()
String[] getLocations()
PermissionInfo[] getPermissions(String)
void setDefaultPermissions(PermissionInfo[])
void setPermissions(String,PermissionInfo[])

class PermissionInfo

PermissionInfo(String,String,String)
PermissionInfo(String)
boolean equals(Object)
final String getActions()
final String getEncoded()
final String getName()

final String getType()
int hashCode()
String toString()

27.0.11 org.osgi.service.prefs

class BackingStoreException extends Exception
BackingStoreException(String)

interface Preferences
String absolutePath()
String[] childrenNames() throws BackingStoreException
void clear() throws BackingStoreException
void flush() throws BackingStoreException
String get(String,String)
boolean getBoolean(String,boolean)
byte[] getByteArray(String,byte[])
double getDouble(String,double)
float getFloat(String,float)
int getInt(String,int)
long getLong(String,long)
String[] keys() throws BackingStoreException
String name()
Preferences node(String)
boolean nodeExists(String) throws BackingStoreException
Preferences parent()
void put(String,String)
void putBoolean(String,boolean)
void putByteArray(String,byte[])
void putDouble(String,double)
void putFloat(String,float)
void putInt(String,int)
void putLong(String,long)
void remove(String)
void removeNode() throws BackingStoreException
void sync() throws BackingStoreException

interface PreferencesService
Preferences getSystemPreferences()
Preferences getUserPreferences(String)
String[] getUsers()

27.0.12 org.osgi.service.provisioning

interface ProvisioningService
static final String MIME_BUNDLE
static final String MIME_BUNDLE_URL
static final String MIME_BYTE_ARRAY
static final String MIME_STRING
static final String PROVISIONING_AGENT_CONFIG
static final String PROVISIONING_REFERENCE
static final String PROVISIONING_ROOTX509
static final String PROVISIONING_RSH_SECRET
static final String PROVISIONING_SPID
static final String PROVISIONING_START_BUNDLE
static final String PROVISIONING_UPDATE_COUNT
void addInformation(Dictionary)
void addInformation(ZipInputStream) throws IOException
Dictionary getInformation()
void setInformation(Dictionary)

27.0.13 org.osgi.service.startlevel

interface StartLevel
int getBundleStartLevel(Bundle)
int getInitialBundleStartLevel()
int getStartLevel()
boolean isBundlePersistentlyStarted(Bundle)
void setBundleStartLevel(Bundle,int)
void setInitialBundleStartLevel(int)
void setStartLevel(int)

27.0.14 org.osgi.service.upnp

interface UPnPAction
String[] getInputArgumentNames()
String getName()
String[] getOutputArgumentNames()
String getReturnArgumentName()
UPnPStateVariable getStateVariable(String)
Dictionary invoke(Dictionary) throws Exception

interface UPnPDevice
static final String CHILDREN_UDN
static final String DEVICE_CATEGORY
static final String FRIENDLY_NAME
static final String ID
static final String MANUFACTURER
static final String MANUFACTURER_URL
static final int MATCH_GENERIC
static final int MATCH_MANUFACTURER_MODEL
static final int MATCH_MANUFACTURER_MODEL_REVISION
static final int MATCH_MANUFACTURER_MODEL_REVISION_SERIAL
static final int MATCH_TYPE
static final String MODEL_DESCRIPTION
static final String MODEL_NAME
static final String MODEL_NUMBER
static final String MODEL_URL
static final String PARENT_UDN
static final String PRESENTATION_URL
static final String SERIAL_NUMBER
static final String TYPE
static final String UDN
static final String UPC
static final String UPNP_EXPORT
Dictionary getDescriptions(String)
UPnPIcon[] getIcons(String)
UPnPService getService(String)

UPnPService[] getServices()

interface UPnPEventListener
static final String UPNP_FILTER
void notifyUPnPEvent(String,String, Dictionary)

interface UPnPIcon
int getDepth()
int getHeight()
InputStream getInputStream() throws IOException
String getMimeType()
int getSize()
int getWidth()

interface UPnPService
static final String ID
static final String TYPE
UPnPAction getAction(String)
UPnPAction[] getActions()
String getId()
UPnPStateVariable getStateVariable(String)
UPnPStateVariable[] getStateVariables()
String getType()
String getVersion()

interface UPnPStateVariable
static final String TYPE_BIN_BASE64
static final String TYPE_BIN_HEX
static final String TYPE_BOOLEAN
static final String TYPE_CHAR
static final String TYPE_DATE
static final String TYPE_DATETIME
static final String TYPE_DATETIME_TZ
static final String TYPE_FIXED_14_4
static final String TYPE_FLOAT
static final String TYPE_I1
static final String TYPE_I2
static final String TYPE_I4
static final String TYPE_INT
static final String TYPE_NUMBER
static final String TYPE_R4
static final String TYPE_R8
static final String TYPE_STRING
static final String TYPE_TIME
static final String TYPE_TIME_TZ
static final String TYPE_UI1
static final String TYPE_UI2
static final String TYPE_UI4
static final String TYPE_URI
static final String TYPE_UUID
String[] getAllowedValues()
Object getDefaultValue()
Class getJavaDataType()
Number getMaximum()
Number getMinimum()
String getName()
Number getStep()
String getUPnPDataType()
boolean sendsEvents()

27.0.15 org.osgi.service.url

abstract class AbstractURLStreamHandlerService extends URLStreamHandler implements URLStreamHandlerService
protected URLStreamHandlerSetter realHandler
AbstractURLStreamHandlerService()
boolean equals(URL,URL)
int getDefaultPort()
InetAddress getHostAddress(URL)
int hashCode(URL)
boolean hostsEqual(URL,URL)
abstract URLConnection openConnection(URL) throws IOException
void parseURL(URLStreamHandlerSetter,URL, String,int,int)
boolean sameFile(URL,URL)
protected void setURL(URL,String,String,int, String,String)
protected void setURL(URL,String,String,int, String,String,String,String)
String toExternalForm(URL)

interface URLConstants
static final String URL_CONTENT_MIMETYPE
static final String URL_HANDLER_PROTOCOL

interface URLStreamHandlerService
boolean equals(URL,URL)
int getDefaultPort()
InetAddress getHostAddress(URL)
int hashCode(URL)
boolean hostsEqual(URL,URL)
URLConnection openConnection(URL) throws IOException
void parseURL(URLStreamHandlerSetter,URL, String,int,int)
boolean sameFile(URL,URL)
String toExternalForm(URL)

interface URLStreamHandlerSetter
void setURL(URL,String,String,int,String, String)
void setURL(URL,String,String,int,String, String,String,String,String)

27.0.16 org.osgi.service.useradmin

interface Authorization
String getName()
String[] getRoles()
boolean hasRole(String)

interface Group extends User
boolean addMember(Role)
boolean addRequiredMember(Role)
Role[] getMembers()
Role[] getRequiredMembers()
boolean removeMember(Role)

interface Role
static final int GROUP
static final int ROLE
static final int USER
String getName()
Dictionary getProperties()
int getType()

interface User extends Role
Dictionary getCredentials()
boolean hasCredential(String,Object)

interface UserAdmin
Role createRole(String,int)
Authorization getAuthorization(User)
Role getRole(String)
Role[] getRoles(String) throws InvalidSyntaxException
User getUser(String,String)
boolean removeRole(String)

class UserAdminEvent
static final int ROLE_CHANGED
static final int ROLE_CREATED
static final int ROLE_REMOVED
UserAdminEvent(ServiceReference,int,Role)
Role getRole()
ServiceReference getServiceReference()
int getType()

interface UserAdminListener
void roleChanged(UserAdminEvent)

final class UserAdminPermission extends BasicPermission
static final String ADMIN
static final String CHANGE_CREDENTIAL
static final String CHANGE_PROPERTY
static final String GET_CREDENTIAL
UserAdminPermission(String,String)
boolean equals(Object)
String getActions()
int hashCode()
boolean implies(Permission)
PermissionCollection newPermissionCollection()
String toString()

27.0.17 org.osgi.service.wireadmin

class BasicEnvelope implements Envelope
BasicEnvelope(Object,Object,String)
Object getIdentification()
String getScope()
Object getValue()

interface Consumer
void producersConnected(Wire[])
void updated(Wire,Object)

interface Envelope
Object getIdentification()
String getScope()
Object getValue()

interface Producer
void consumersConnected(Wire[])
Object polled(Wire)

interface Wire
Class[] getFlavors()
Object getLastValue()
Dictionary getProperties()
String[] getScope()
boolean hasScope(String)
boolean isConnected()
boolean isValid()
Object poll()
void update(Object)

interface WireAdmin
Wire createWire(String,String,Dictionary)
void deleteWire(Wire)
Wire[] getWires(String) throws InvalidSyntaxException
void updateWire(Wire,Dictionary)

class WireAdminEvent
static final int CONSUMER_EXCEPTION
static final int PRODUCER_EXCEPTION
static final int WIRE_CONNECTED
static final int WIRE_CREATED
static final int WIRE_DELETED
static final int WIRE_DISCONNECTED
static final int WIRE_TRACE
static final int WIRE_UPDATED
WireAdminEvent(ServiceReference,int,Wire,Throwable)
ServiceReference getServiceReference()
Throwable getThrowable()
int getType()
Wire getWire()

interface WireAdminListener
void wireAdminEvent(WireAdminEvent)

interface WireConstants
static final String WIREADMIN_CONSUMER_COMPOSITE
static final String WIREADMIN_CONSUMER_FLAVORS
static final String WIREADMIN_CONSUMER_PID
static final String WIREADMIN_CONSUMER_SCOPE
static final String WIREADMIN_EVENTS
static final String WIREADMIN_FILTER
static final String WIREADMIN_PID
static final String WIREADMIN_PRODUCER_COMPOSITE
static final String WIREADMIN_PRODUCER_FILTERS
static final String WIREADMIN_PRODUCER_FLAVORS
static final String WIREADMIN_PRODUCER_PID
static final String WIREADMIN_PRODUCER_SCOPE
static final String WIREADMIN_SCOPE_ALL

static final String WIREVALUE_CURRENT
static final String WIREVALUE_DELTA_ABSOLUTE
static final String WIREVALUE_DELTA_RELATIVE
static final String WIREVALUE_ELAPSED
static final String WIREVALUE_PREVIOUS

final class WirePermission extends BasicPermission
static final String CONSUME
static final String PRODUCE
WirePermission(String,String)
boolean equals(Object)
String getActions()
int hashCode()
boolean implies(Permission)
PermissionCollection newPermissionCollection()
String toString()

27.0.18 org.osgi.util.measurement

class Measurement implements Comparable
Measurement(double,double,Unit,long)
Measurement(double,double,Unit)
Measurement(double,Unit)
Measurement(double)
Measurement add(Measurement)
Measurement add(double,Unit)
Measurement add(double)
int compareTo(Object)
Measurement div(Measurement)
Measurement div(double,Unit)
Measurement div(double)
boolean equals(Object)
final double getError()
final long getTime()
final Unit getUnit()
final double getValue()
int hashCode()
Measurement mul(Measurement)
Measurement mul(double,Unit)
Measurement mul(double)
Measurement sub(Measurement)
Measurement sub(double,Unit)
Measurement sub(double)
String toString()

class State
State(int,String,long)
State(int,String)
boolean equals(Object)
final String getName()
final long getTime()
final int getValue()
int hashCode()
String toString()

class Unit
static final Unit A
static final Unit C
static final Unit cd
static final Unit F
static final Unit Gy
static final Unit Hz
static final Unit J
static final Unit K
static final Unit kat
static final Unit kg
static final Unit lx
static final Unit m
static final Unit m2
static final Unit m3
static final Unit m_s
static final Unit m_s2
static final Unit mol
static final Unit N
static final Unit Ohm
static final Unit Pa
static final Unit rad
static final Unit S
static final Unit s
static final Unit T
static final Unit unity
static final Unit V
static final Unit W
static final Unit Wb
boolean equals(Object)
int hashCode()
String toString()

27.0.19 org.osgi.util.position

class Position
Position(Measurement,Measurement, Measurement,Measurement, Measurement)
Measurement getAltitude()
Measurement getLatitude()
Measurement getLongitude()
Measurement getSpeed()
Measurement getTrack()

27.0.20 org.osgi.util.tracker

class ServiceTracker implements ServiceTrackerCustomizer
protected final BundleContext context
protected final Filter filter
ServiceTracker(BundleContext, ServiceReference, ServiceTrackerCustomizer)
ServiceTracker(BundleContext,String, ServiceTrackerCustomizer)
ServiceTracker(BundleContext,Filter, ServiceTrackerCustomizer)
Object addingService(ServiceReference)
void close()
protected void finalize() throws Throwable
Object getService(ServiceReference)
Object getService()
ServiceReference getServiceReference()
ServiceReference[] getServiceReferences()
Object[] getServices()
int getTrackingCount()
void modifiedService(ServiceReference, Object)
void open()
void remove(ServiceReference)
void removedService(ServiceReference, Object)

int size()
Object waitForService(long) throws InterruptedException

interface ServiceTrackerCustomizer

Object addingService(ServiceReference)
void modifiedService(ServiceReference, Object)
void removedService(ServiceReference, Object)

27.0.21 org.osgi.util.xml

class XMLParserActivator implements BundleActivator , ServiceFactory

static final String DOMCLASSFILE
static final String DOMFACTORYNAME
static final String PARSER_NAMESPACEAWARE
static final String PARSER_VALIDATING
static final String SAXCLASSFILE
static final String SAXFACTORYNAME
XMLParserActivator()
Object getService(Bundle, ServiceRegistration)
void setDOMProperties(DocumentBuilderFactory,Hashtable)
void setSAXProperties(SAXParserFactory, Hashtable)
void start(BundleContext) throws Exception
void stop(BundleContext) throws Exception
void ungetService(Bundle, ServiceRegistration,Object)

27.0.22

Index

A

B

C

D

E

F

G

J

N

O

P

S

T

U

End Of Document