OSGi Service Platform, Core Specification Release 4

OSGi Service Platform Core Specification

The OSGi Alliance

Release 4
August 2005

IOS Press

Ohmsha

Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

Table Of Contents

1 **Introduction** 2

1.1 Sections 2

1.2 What is New 2

1.3 Reader Level 2

1.4 Conventions and Terms 3

1.5 The Specification Process 6

1.6 Version Information 7

1.7 Compliance Program 8

1.8 References 9

2 **Framework Specification** 11

2.1 Introduction 11

2.2 Bundles 14

2.3 Manifest Headers 15

2.4 The Bundle Name-space 17

2.5 Execution Environment 24

2.6 Loading Native Code Libraries 26

2.7 Finding Classes and Resources 28

2.8 The Bundle Object 29

2.9 The Bundle Context 34

2.10 Services 38

2.11 Stale References 44

2.12 Filters 45

2.13 Service Factories 47

2.14 Importing and Exporting Services 48

2.15 Releasing Services 48

2.16 Unregistering Services 49

2.17 Configurable Services 49

2.18 Events 50

2.19 Framework Startup and Shutdown 52

2.20 Security 53

2.21 The Framework on Java 1.1 56

2.22 Changes 57

2.23 org.osgi.framework 59

2.24 References 110

3 **Package Admin Service Specification** 113

3.1 Introduction 113

3.2 Package Admin 114

3.3 Security 115
3.4 Changes 115
3.5 org.osgi.service.packageadmin 115

4 **Start Level Service Specification** **123**

4.1 Introduction 123
4.2 Start Level Service 124
4.3 Compatibility Mode 128
4.4 Example Applications 128
4.5 Security 129
4.6 org.osgi.service.startlevel 129

5 **Permission Admin Service Specification** **135**

5.1 Introduction 135
5.2 Permission Admin service 136
5.3 Security 138
5.4 Changes 138
5.5 org.osgi.service.permissionadmin 138

6 **Conditional Permission Admin Specification** **143**

6.1 Introduction 143
6.2 The Conditional Permission Admin Service 143
6.3 Security 143
6.4 org.osgi.service.condpermadmin 143

7 **URL Handlers Service Specification** **145**

7.1 Introduction 145
7.2 Factories in java.net 148
7.3 Framework Procedures 149
7.4 Providing a New Scheme 153
7.5 Providing a Content Handler 154
7.6 Security Considerations 154
7.7 org.osgi.service.url 155
7.8 References 158

8 **Service Tracker Specification** **161**

8.1 Introduction 161
8.2 ServiceTracker Class 162
8.3 Using a Service Tracker 163
8.4 Customizing the ServiceTracker class 163
8.5 Customizing Example 164
8.6 Security 165
8.7 Changes 165

8.8 org.osgi.util.tracker 165

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

Publisher

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

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

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

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

Distributor in Japan

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

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

Foreword

John Barr, *President OSGi*

1 Introduction

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

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

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

1.1 Sections

1.2 What is New

1.3 Reader Level

This specification is written for the following audiences:

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

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

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

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

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

1.4 Conventions and Terms

1.4.1 Typography

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

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

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

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

is equivalent to:

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

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

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

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

Spaces are ignored unless specifically noted.

1.4.2 Object Oriented Terminology

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

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

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

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

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

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

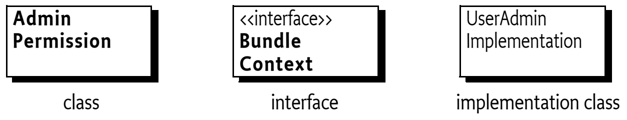
Service objects are registered with the OSGi Framework. Registration consists of the service object, a set of properties, and a list of classes and interfaces implemented by this service object. The classes and interfaces are used for type safety *and* naming. Therefore, it is said that a service object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

1.4.3 Diagrams

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

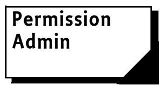
Classes or interfaces are depicted as rectangles, as in Figure 1. Interfaces are indicated with the qualifier <<interface>> as the first line. The name of the class/interface is indicated in bold when it is part of the specification. Implementation classes are sometimes shown to demonstrate a possible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

Figure 1 *Class and interface symbol*



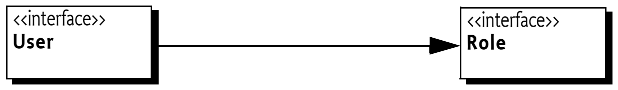
If an interface or class is used as a service object, it will have a black triangle in the bottom right corner.

Figure 2 *Service symbol*



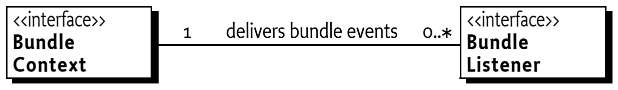
Inheritance (the extends or implements keyword in Java class definitions) is indicated with an arrow. Figure 3 shows that User implements or extends Role.

Figure 3 *Inheritance (implements or extends) symbol*



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

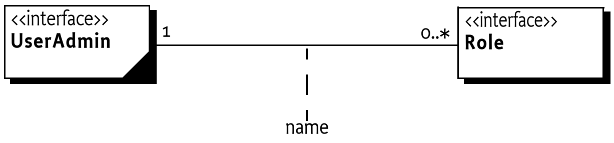
Figure 4 *Relations symbol*



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

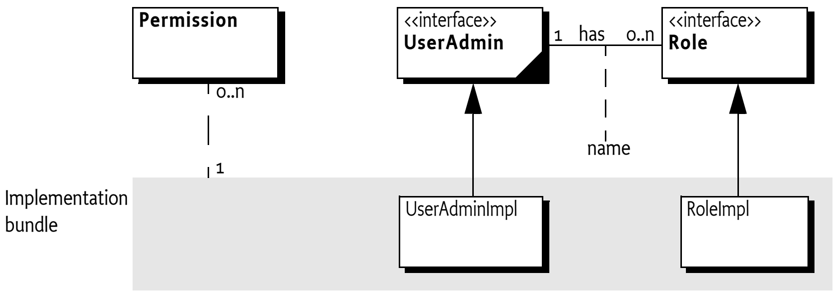
When a relationship is qualified by a name or an object, it is indicated by drawing a dotted line perpendicular to the relation and connecting this line to a class box or a description. Figure 5 shows that the relationship between a UserAdmin class and a Role class is qualified by a name. Such an association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

Figure 5 *Associations symbol*



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

Figure 6 *Bundles*



1.4.4 Key Words

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

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

1.5 The Specification Process

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

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

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

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

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

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

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

1.6 Version Information

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

New for this specification are:

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

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

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

Table 1 Packages and versions

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

1.7 Compliance Program

The OSGi offers a compliance program for the software product that includes an OSGi Framework and a set of zero or more core bundles collectively referred to as a Service Platform. Any services which exist in the org.osgi name-space and that are offered as part of a Service Platform must pass the conformance test suite in order for the product to be considered for inclusion in the compliance program. A Service Platform may be tested in isolation and is independent of its host Virtual Machine. Certification means that a product has passed the conformance test suite(s) and meets certain criteria necessary for admission to the program, including the requirement for the supplier to warrant and represent that the product conforms to the applicable OSGi specifications, as defined in the compliance requirements.

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

1.7.1 Compliance Claims.

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

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

1.8 References

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

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

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

2 Framework Specification

Version 1.3

2.1 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.

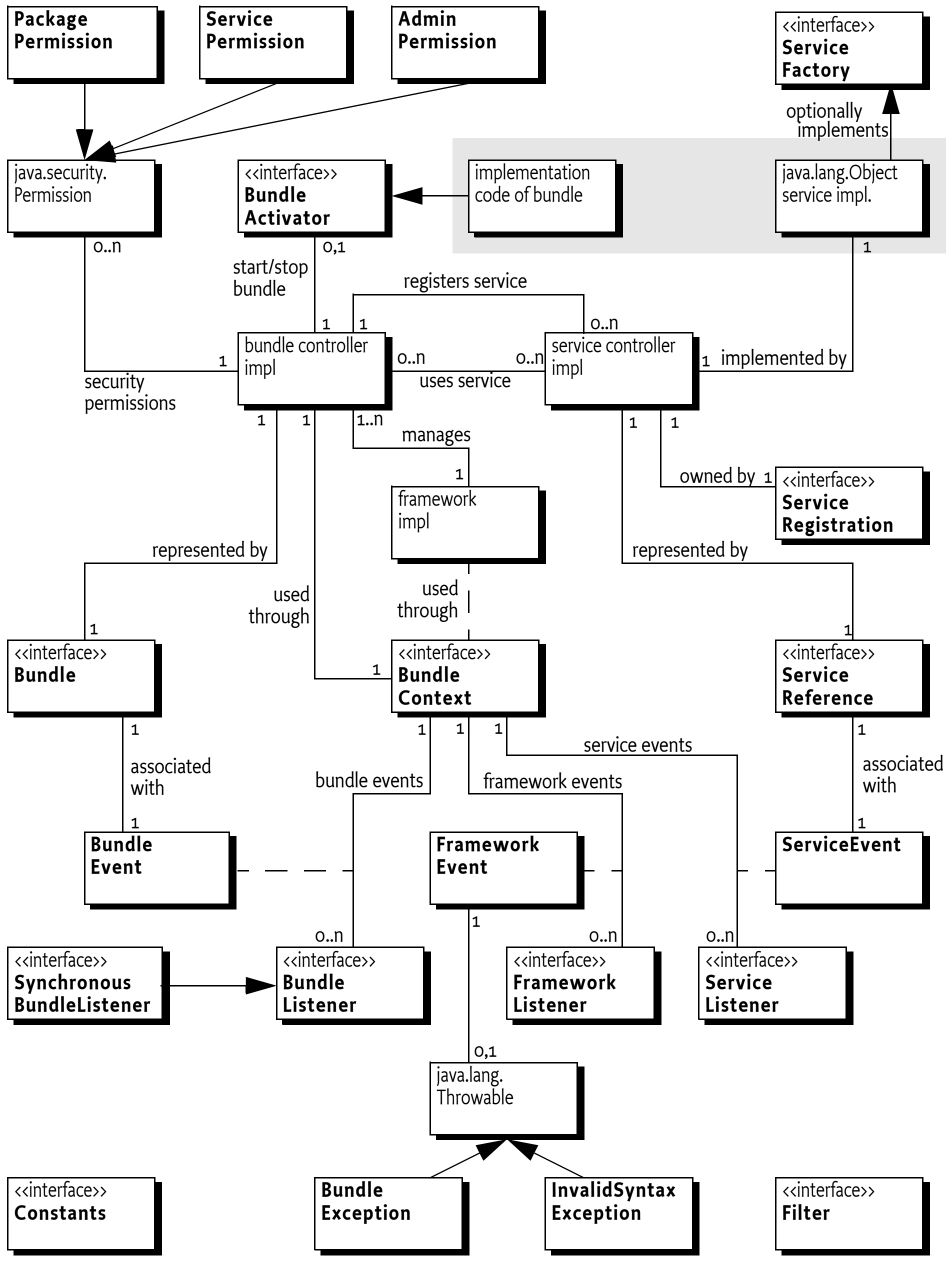
The Framework allows bundles to select an available implementation at run-time through the Framework service registry. Bundles register new services, receive notifications about the state of services, or look up existing services to adapt to the current capabilities of the device. This aspect of the 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.

2.1.1 Entities

Figure 7 on page 13 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 7 *Class Diagram* org.osgi.framework



2.2 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 18.
- 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.

2.2.1 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 52 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).

2.3 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 [12] *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 16.

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.

2.3.1 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.

2.3.2 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 31.
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 23.
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 26.
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 24.
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 21.

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 19.
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 20

Table 2 *Manifest Headers*

2.4 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.

2.4.1 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 23 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.

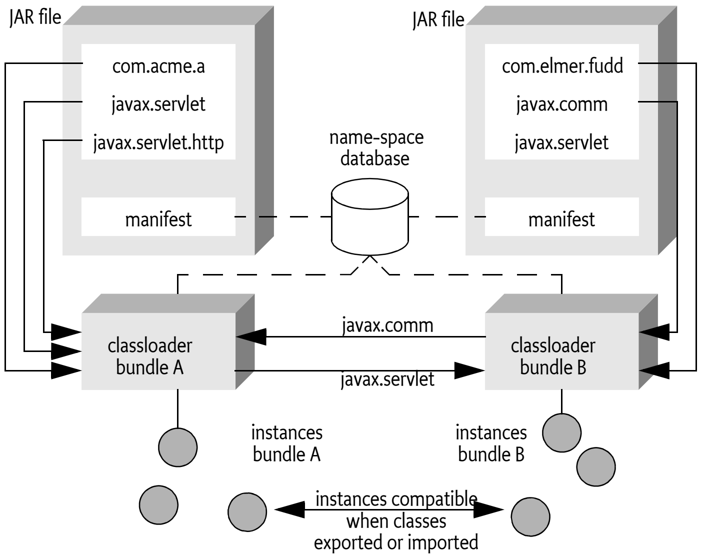
The classloader for a bundle is created between installing and resolving the bundle.

2.4.2 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 8 *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.

2.4.3 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 20.

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 [10] *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"
```

2.4.4 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>]. The Framework must *not* throw a SecurityException when a package is imported without the appropriate PackagePermission, only the visibility of that package is affected and the bundle can thus not resolve. 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 [10] *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"
```

2.4.5 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 hiding packages where the bundle does not have the necessary PackagePermission[IMPORT|EXPORT,<package>] 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.

2.4.5.1 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.

2.4.5.2 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.

2.4.6 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.

2.4.7 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.

2.4.8 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.

2.4.9 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 ::= jar-path  |  '.'
```

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.

2.5 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.

Bundles should list all (known) execution environments that can run the bundle.

2.5.1 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.

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.profiles
```

For example, Foundation Profile has an Execution Environment name of CDC-1.0/Foundation-1.0. The structure of the name is should follow the following rules:

```
ee-name = [ <configuration> '-' <version> '/' ]
            <profile> '-' <version>
```

Configuration and profile names could be defined by JCP or OSGi. If an execution environment does not have a configuration or profile, the profile part is the name identifying the execution environment. These are guidelines and not normative.

Table 3 on page 25, contains a number of examples of the most common execution environments.

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.
JRE-1.1	Java 1.1.x
J2SE-1.2	Java 2 SE 1.2.x
J2SE-1.3	Java 2 SE 1.3.x
J2SE-1.4	Java 2 SE 1.4.x
PersonalJava-1.1	Personal Java 1.1
PersonalJava-1.2	Personal Java 1.2
CDC-1.0/PersonalBasis-1.0	J2ME Personal Basis Profile
CDC-1.0/PersonalJava-1.0	J2ME Personal Java Profile

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. This property is defined as *volatile*. A Framework implementation must not cache this information because bundles may change this system property at any time. The purpose of this volatility is testing and possible extending the execution environments at run-time.

A Framework should list all execution environments that are available in its Java VM for the org.osgi.framework.executionenvironment property.

2.6 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  ::=  jar-path ( ';' jar-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.

2.6.1 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 35). 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.

2.7 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 23.

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

2.7.1 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.

2.7.2 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.

According to the documentation of the ClassLoader class, packages beginning with java may only be loaded through the bootstrap class loader. However, the text is not completely clear and seems to allow loading java classes with a null name parameter. The reason that this specification still allows loading java packages is that an OSGi Service Platform has PackagePermission that allows the Operator to securely control the environment.

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..

2.8 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.

2.8.1 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.

2.8.2 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.

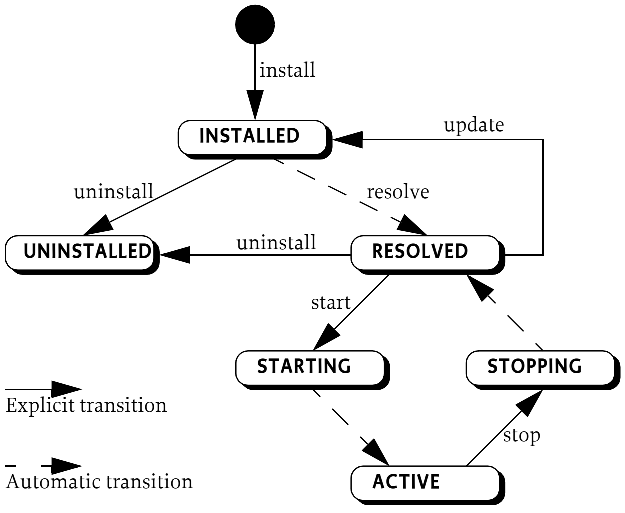
A location string uniquely identifies a bundle and must not change when a bundle is updated.

2.8.3 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 9 *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 123.

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)
   ...
```

2.8.4 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 26.

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.

2.8.5 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.

2.8.6 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 123.

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.

2.8.7 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.

2.8.8 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 [7] *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.

2.8.9 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.

If a new bundle is installed it must use the currently exported packages even if they refer to the packages of an uninstalled bundle, unless the uninstalled bundle was the only user of that package.

2.9 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 31.
- Interrogating other bundles installed in the OSGi environment. See *Getting Bundle Information* on page 34.
- Obtaining a persistent storage area. See *Persistent Storage* on page 35.
- Retrieving service objects of registered services. See *ServiceReference Objects* on page 38.
- Registering services in the Framework service. See *Registering Services* on page 39.
- Subscribing or unsubscribing to events broadcast by the Framework. See *Events* on page 50.

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.

2.9.1 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.

2.9.2 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.

2.9.3 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 [11] *Codes for the Representation of Names of Languages* for valid values.

Table 4 Property Names

Property name	Description
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.
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
68k		68000 and up
ARM		Intel Strong ARM
Alpha		Compaq
Ignite	psc1k	PTSC
Mips		SGI
PArisc		Hewlett Packard
PowerPC	power ppc	Motorola/IBM
Sparc		SUN
x86	pentium i386 i486 i586 i686	Intel

Name	Aliases	Description
AIX		IBM
DigitalUnix		Compaq
FreeBSD		Free BSD

Table 4 Property Names

Property name	Description		
	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
	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.

2.10 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.

2.10.1 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 41.

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.

2.10.2 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.

2.10.3 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 47.

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 18. 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.

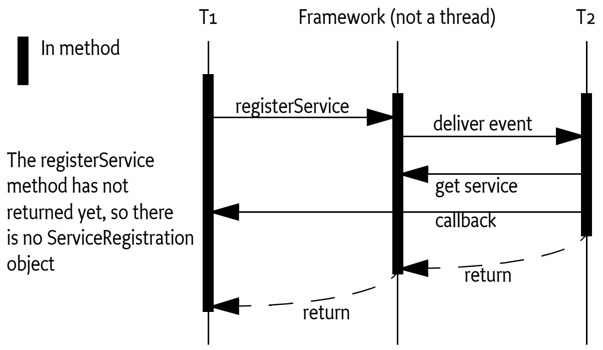
If the service object is successfully registered, the Framework must return a ServiceRegistration object to the caller. A service object can be unregistered only by the holder of its ServiceRegistration object (see the unregister() method). Every successful service object registration must yield a unique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

2.10.4 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 10 on page 40 shows such a sequence.

Figure 10 *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.

2.10.5 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 45 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.

2.10.6 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 55 for more information.

2.10.7 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

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(Service Reference), 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.
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.

Table 5 Standard Framework Service Registry Properties

Property Key	Type	Constants	Property Description
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*

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 45.

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.

2.10.8 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.

2.10.9 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 47 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.

2.10.10 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.

2.11 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 161.

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 47.
- 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.

2.12 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 [8] *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 [8] *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.

2.13 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 39.

- 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.

2.14 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>
```

2.15 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.

2.16 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.

2.17 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 1.

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.

2.18 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.

2.18.1 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 161).

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.

The Framework must publish a `FrameworkEvent.ERROR` if a callback to an event listener generates an unchecked exception, except when the callback happens while delivering a `FrameworkEvent.ERROR` (to prevent an inifinite loop).

2.18.2 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.

2.18.3 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.

2.19 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.

2.19.1 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 50.
2. The system bundle enters the STARTING state. More information about the system bundle can be found in *The System Bundle* on page 14.
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 wrapped in a BundleException and then published as a Framework event of type FrameworkEvent.ERROR. Bundles and their different states are discussed in *The Bundle Object* on page 29. If the Framework implements the optional Start Level specification, this behavior is different. See *Start Level Service Specification* on page 123.
4. The system bundle enters the ACTIVE state.
5. A Framework event of type FrameworkEvent.STARTED is broadcast.

2.19.2 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 14. 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 wrapped in a BundleException and then 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 123.
3. Event handling is disabled.

2.20 Security

The Framework security model is based on the Java 2 specification. If security checks are performed, they must be done according to [9] *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.

2.20.1 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 [9] *The Java Security Architecture for JDK 1.2*.

2.20.2 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.

2.20.3 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.

2.20.4 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.

2.20.5 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 Bundle Context.registerService, the registering bundle must have the ServicePermission to register all the named classes. See *Registering Services* on page 39.

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 38.

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.

2.20.6 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.

2.20.7 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.

2.21 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.

2.21.1 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.

2.21.2 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.

2.21.3 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.

2.21.4 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.

2.22 Changes

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

2.22.1 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 21.

2.22.2 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 29.

2.22.3 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 27. A common mistake with the native code clause was also highlighted with an example.

2.22.4 Synchronization Pitfalls

Using synchronized with Framework callbacks can cause deadlocks. A section has been added that discusses these issues. See *Synchronization Pitfalls* on page 51.

2.22.5 New Constants

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

2.22.6 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 35.

2.22.7 Use of System Properties

The mapping of System properties to the BundleContext.getProperty method has been clarified in *Environment Properties* on page 35.

2.22.8 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.

2.22.9 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.

2.22.10 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 53.

2.22.11 Bundle-RequiredExecutionEnvironment

A new manifest header is introduced to specify a bundle's requirements on the execution environment. This is explained in *Execution Environment* on page 24.

2.22.12 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 45 for more information.

2.22.13 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.

2.22.14 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 40.

2.22.15 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[].

2.23 org.osgi.framework

The OSGi Framework Package. Specification Version 1.3.

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=1.3
```

2.23.1 Summary

- AdminPermission - Indicates the caller's authority to perform lifecycle operations on or to get sensitive information about a bundle. [p.60]
- Bundle - An installed bundle in the Framework. [p.61]
- BundleActivator - Customizes the starting and stopping of this bundle. [p.72]

- BundleContext - A bundle's execution context within the Framework. [p.73]
- BundleEvent - A Framework event describing a bundle lifecycle change. [p.82]
- BundleException - A Framework exception used to indicate that a bundle lifecycle problem occurred. [p.15]
- BundleListener - A `BundleEvent` listener. [p.84]
- Configurable - Supports a configuration object. [p.85]
- Constants - Defines standard names for the OSGi environment property, service property, and Manifest header attribute keys. [p.85]
- Filter - An RFC 1960-based Filter. [p.98]
- FrameworkEvent - A general Framework event. [p.99]
- FrameworkListener - A `FrameworkEvent` listener. [p.100]
- InvalidSyntaxException - A Framework exception. [p.101]
- PackagePermission - A bundle's authority to import or export a package. [p.101]
- ServiceEvent - A service lifecycle change event. [p.103]
- ServiceFactory - Allows services to provide customized service objects in the OSGi environment. [p.104]
- ServiceListener - A `ServiceEvent` listener. [p.105]
- ServicePermission - Indicates a bundle's authority to register or get a service. [p.106]
- ServiceReference - A reference to a service. [p.107]
- ServiceRegistration - A registered service. [p.108]
- SynchronousBundleListener - A synchronous `BundleEvent` listener. [p.110]

2.23.2 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`.

2.23.2.1 public AdminPermission( )

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

2.23.2.2 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.

2.23.2.3 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.

2.23.2.4 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.

2.23.2.5 public PermissionCollection newPermissionCollection( )

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

Returns A new `PermissionCollection` object.

2.23.3 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.62]
- `INSTALLED`[p.62]
- `RESOLVED`[p.62]
- `STARTING`[p.62]
- `STOPPING`[p.62]
- `ACTIVE`[p.61]

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 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.

2.23.3.1 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.

2.23.3.2 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.

2.23.3.3 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.86] Manifest header.
- The bundle's package dependencies from its Constants.EXPORT_PACKAGE[p.90] and Constants.IMPORT_PACKAGE[p.93] 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.

2.23.3.4 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.67] method is active. A bundle will be in this state when the bundle's BundleActivator.start[p.72] 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.

2.23.3.5 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.68] method is active. A bundle will be in this state when the bundle's BundleActivator.stop[p.72] method is called. When this method completes the bundle is stopped and will move to the RESOLVED state.

The value of STOPPING is 0x00000010.

2.23.3.6 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 but references to the `Bundle` object may still be available and used for introspection.

The value of `UNINSTALLED` is 0x00000001.

2.23.3.7 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.

2.23.3.8 public URL getEntry(String name)

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

□ Returns a URL to the specified entry in this bundle. The bundle's classloader is not used to search for the specified entry. Only the contents of the bundle is searched for the specified entry. A specified path of "/" indicates the root of the bundle.

This method returns a URL to the specified entry, or `null` if the entry could not be found or if the caller does not have the `AdminPermission` and the Java Runtime Environment supports permissions.

Returns A URL to the specified entry, or `null` if the entry 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 1.3

2.23.3.9 public Enumeration getEntryPaths(String path)

path the path name to get the entry path names for.

□ Returns enumeration of all the paths to entries within the bundle whose longest sub-path matches the supplied path argument. The bundle's classloader is not used to search for entries. Only the contents of the bundle is searched. A specified path of "/" indicates the root of the bundle.

Returned paths indicating subdirectory paths end with a "/". The returned paths are all relative to the root of the bundle and have a leading "/".

This method returns null if no entries could be found that match the specified path or if the caller does not have AdminPermission and the Java Runtime Environment supports permissions.

Returns An Enumeration of the entry paths that are contained in the specified path 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 1.3

2.23.3.10 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. If a Manifest header value starts with "%", it will be localized with the localization properties file for the default locale.

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.

See Also Constants.BUNDLE_LOCALIZATION[p.86]

2.23.3.11 public Dictionary getHeaders(String localeString)

□ Returns this bundle's Manifest headers and values localized to the specifed locale.

This method performs the same function as Bundle.getHeaders() except the manifest header values are localized to the specified locale. If a Manifest header value starts with "%", it will be localized with the localization properties file for the specified locale. If null is specified as the locale string, the header values will be localized using the default locale. If the empty string ("") is specified as the locale string, the header values will not be localized but any leading "%"s will be stripped off of the header values.

This method will continue to return Manifest header information while this bundle is in the UNINSTALLED state, however the header values will only be localized to the default locale.

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.

See Also `getHeaders()`[p.64], `Constants.BUNDLE_LOCALIZATION`[p.86]

Since 1.3

2.23.3.12 public String getLocation( )

□ Returns this bundle's location identifier.

The bundle location identifier is the location passed to `BundleContext.installBundle` 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.

2.23.3.13 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.108], `ServiceReference`[p.107], `ServicePermission`[p.106]

2.23.3.14 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

2.23.3.15 public Enumeration getResources(String name)

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

□ Find the specified resources 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 an Enumeration of URLs to the named resources, 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 1.3

2.23.3.16 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.107], `ServicePermission`[p.106]

2.23.3.17 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`.

2.23.3.18 public String getSymbolicName( )

□ Returns the symbolic name of this bundle as specified by its `Bundle-SymbolicName` manifest header. The name must be unique, it is recommended to use a reverse domain name naming convention like that used for java packages. If the bundle does not have a specified symbolic name then `null` is returned.

This method will continue to return this bundle's symbolic name while this bundle is in the `UNINSTALLED` state.

Returns The symbolic name of this bundle.

Since 1.3

2.23.3.19 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 `instanceofjava.security.Permission`.

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

2.23.3.20 public Class loadClass(String name) throws ClassNotFoundException

name The name of the class to load.

□ Loads the specified class using this bundle's classloader.

If this bundle's state is `INSTALLED`, this method will attempt to resolve the bundle before attempting to load the class.

If the bundle cannot be resolved, a Framework event of type `FrameworkEvent.ERROR`[p.99] is broadcast containing a `BundleException` with details of the reason the bundle could not be resolved. This method must then throw a `ClassNotFoundException`.

If this bundle's state is `UNINSTALLED`, then an `IllegalStateException` is thrown.

Returns The Class object for the requested class.

Throws `ClassNotFoundException` – If no such class can be found or if the caller does not have the `AdminPermission`, and the Java Runtime Environment supports permissions.

`IllegalStateException` – If this bundle has been uninstalled.

Since 1.3

2.23.3.21 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.72] 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 Persistently record 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.82] 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 `AdminPermission`, and the Java Runtime Environment supports permissions.

2.23.3.22 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 Persistently record that this bundle has been stopped. When the Framework is restarted, this bundle will not be automatically started.
4 If this bundle's state is not ACTIVE then this method returns immediately.
5 This bundle's state is set to STOPPING.
6 The BundleActivator.stop[p.72] 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.83] 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.

2.23.3.23 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.99] is broadcast containing the exception.
3 This bundle's state is set to `UNINSTALLED`.
4 A bundle event of type `BundleEvent.UNINSTALLED`[p.83] 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.68]

2.23.3.24 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.89] 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.83] 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.99] 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.68], `start()`[p.67]

2.23.3.25 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.70]

2.23.4 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()`.

2.23.4.1 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.67]

2.23.4.2 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.68]

2.23.5 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.61] 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.72] method. The same BundleContext object will be passed to this bundle when it is stopped using the BundleActivator.stop[p.72] 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 ServiceListener 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.

2.23.5.1 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.75] 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.82], BundleListener[p.84]

2.23.5.2 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.75] 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.99], `FrameworkListener`[p.100]

2.23.5.3 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.75] for a definition of context bundle, and `Filter`[p.98] 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.94] property. If `filter` is `null`, all services are considered to match the filter.

  When using a `filter`, it is possible that the `ServiceEvent`s 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.103], `ServiceListener`[p.105], `ServicePermission`[p.106]

2.23.5.4 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.74]

2.23.5.5 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.98] for a description of the filter string syntax.

If the filter cannot be parsed, an `InvalidSyntaxException`[p.101] 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

2.23.5.6 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.

2.23.5.7 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.

2.23.5.8 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.

2.23.5.9 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.75] 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.

2.23.5.10 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.92] - The OSGi Framework version.
- `Constants.FRAMEWORK_VENDOR`[p.92] - The Framework implementation vendor.
- `Constants.FRAMEWORK_LANGUAGE`[p.91] - The language being used. See ISO 639 for possible values.
- `Constants.FRAMEWORK_OS_NAME`[p.92] - The host computer operating system.
- `Constants.FRAMEWORK_OS_VERSION`[p.92] - The host computer operating system version number.
- `Constants.FRAMEWORK_PROCESSOR`[p.92] - 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.87] `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.

2.23.5.11 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.76] the context bundle's use count for that service is incremented by one. Each time the service is released by `ungetService`[p.81] 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.75] 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.105] 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.99] 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.81] , `ServiceFactory`[p.104]

2.23.5.12 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.78] 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.96] 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.95] 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.78]

2.23.5.13 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.98] 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.101] 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.94] 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.

2.23.5.14 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.15] 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 A bundle event of type `BundleEvent.INSTALLED`[p.82] is broadcast.
8 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.

2.23.5.15 public Bundle installBundle(String location, InputStream input) throws BundleException

location The location identifier of the bundle to install.

input 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.78]

2.23.5.16 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.94] .

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.85] 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.109] 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.75] for a definition of context bundle. Other bundles can locate the service by using either the getServiceReferences[p.78] or getServiceReference[p.77] method.

A bundle can register a service object that implements the ServiceFactory[p.104] 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.95] identifying the registration number of the service, and a property named Constants.OBJECTCLASS[p.94] 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.103] 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.108], ServiceFactory[p.104]

2.23.5.17 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.80] 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.94] 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.80]

2.23.5.18 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.75] 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.

2.23.5.19 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.75] 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.

2.23.5.20 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.75] 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.

2.23.5.21 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.75] 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.105] 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.76], `ServiceFactory`[p.104]

2.23.6 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 Alliance reserves the right to extend the set of types.

2.23.6.1 public static final int INSTALLED = 1

The bundle has been installed.

The value of `INSTALLED` is 0x00000001.

See Also `BundleContext.installBundle(String)`[p.78]

2.23.6.2 public static final int RESOLVED = 32

The bundle has been resolved.

The value of `RESOLVED` is 0x00000020.

See Also `Bundle.RESOLVED`[p.62]

Since 1.3

2.23.6.3 public static final int STARTED = 2

The bundle has been started.

The value of STARTED is 0x00000002.

See Also Bundle.start[p.67]

2.23.6.4 public static final int STOPPED = 4

The bundle has been stopped.

The value of STOPPED is 0x00000004.

See Also Bundle.stop[p.68]

2.23.6.5 public static final int UNINSTALLED = 16

The bundle has been uninstalled.

The value of UNINSTALLED is 0x00000010.

See Also Bundle.uninstall[p.69]

2.23.6.6 public static final int UNRESOLVED = 64

The bundle has been unresolved.

The value of UNRESOLVED is 0x00000040.

See Also Bundle.INSTALLED[p.62]

Since 1.3

2.23.6.7 public static final int UPDATED = 8

The bundle has been updated.

The value of UPDATED is 0x00000008.

See Also Bundle.update()[p.70]

2.23.6.8 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.

2.23.6.9 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.

2.23.6.10 public int getType( )

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

- INSTALLED[p.82]
- STARTED[p.82]
- STOPPED[p.83]
- UPDATED[p.83]
- UNINSTALLED[p.83]
- RESOLVED[p.82]
- UNRESOLVED[p.83]

Returns The type of lifecycle event.

2.23.7 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.

This exception is updated to conform to the general purpose exception chaining mechanism.

2.23.7.1 public BundleException(String msg, Throwable cause)

msg The associated message.

cause The cause of this exception.

□ Creates a `BundleException` that wraps another exception.

2.23.7.2 public BundleException(String msg)

msg The message.

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

2.23.7.3 public Throwable getCause( )

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

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

Since 1.3

2.23.7.4 public Throwable getNestedException( )

□ Returns any nested exceptions included in this exception.

This method predates the general purpose exception chaining mechanism. The `getCause()`[p.84] method is now the preferred means of obtaining this information.

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

2.23.7.5 public Throwable initCause(Throwable cause)

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

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

Since 1.3

2.23.8 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.73] method. `BundleListeners` are called with a `BundleEvent` object when a bundle has been installed, started, stopped, updated, or uninstalled.

See Also `BundleEvent`[p.82]

2.23.8.1 public void bundleChanged(BundleEvent event)

event The `BundleEvent`.

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

2.23.9 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`.

Deprecated Please use the Configuration Admin

2.23.9.1 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.60] 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.

Deprecated Please use the Configuration Admin

2.23.10 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.64], `BundleContext.getProperty`[p.76], `BundleContext.registerService(String[],Object,Dictionary)`[p.80]

Since 1.1

2.23.10.1 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.

2.23.10.2 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.

2.23.10.3 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.

2.23.10.4 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.

2.23.10.5 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.

2.23.10.6 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.

2.23.10.7 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.

2.23.10.8 public static final String BUNDLE_LOCALIZATION = "Bundle-

Localization"

Manifest header (named "Bundle-Localization") identifying the base name of the bundle's localization file.

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

Since 1.3

2.23.10.9 public static final String BUNDLE_LOCALIZATION_DEFAULT_BASENAME = "META-INF/bundle"

Default value for the Bundle-Localization manifest header.

See Also `BUNDLE_LOCALIZATION`[p.86]

Since 1.3

2.23.10.10 public static final String BUNDLE_MANIFESTVERSION = "Bundle-ManifestVersion"

Manifest header (named "Bundle-ManifestVersion") identifying the bundle manifest version. A bundle manifest may express the version of the syntax in which it is written by specifying a bundle manifest version. Bundles exploiting OSGi R4, or later, syntax must specify a bundle manifest version.

The bundle manifest version defined by OSGi R4 or, more specifically, by V1.3 of the OSGi Framework Specification is "2".

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

Since 1.3

2.23.10.11 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.

2.23.10.12 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.

2.23.10.13 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 ...
```

2.23.10.14 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 ...
```

2.23.10.15 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" ...
```

2.23.10.16 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 ...
```

2.23.10.17 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

2.23.10.18 public static final String BUNDLE_SYMBOLICNAME = "Bundle-SymbolicName"

Manifest header (named "Bundle-SymbolicName") identifying the bundle's symbolic name.

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

Since 1.3

2.23.10.19 public static final String BUNDLE_SYMBOLICNAME_ATTRIBUTE =

"bundle-symbolic-name"

Manifest header attribute (named "bundle-symbolic-name") identifying the symbolic name of a bundle which exports a package specified in the Import-Package manifest header.

The attribute value is encoded in the Import-Package manifest header like:

```
 Import-Package: org.osgi.framework; bundle-symbolic-
name="com.acme.module.test"
```

Since 1.3

2.23.10.20 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.

2.23.10.21 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.

2.23.10.22 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.

2.23.10.23 public static final String BUNDLE_VERSION_ATTRIBUTE = "bundle-version"

Manifest header attribute (named "bundle-version") identifying a range of versions for a bundle specified in the Require-Bundle or Fragment-Host manifest headers. The default value is `0.0.0`.

The attribute value is encoded in the Require-Bundle manifest header like:

```
 Require-Bundle: com.acme.module.test; bundle-version="1.1"
 Require-Bundle: com.acme.module.test; bundle-version="[1.0,
2.0)"
```

The bundle-version attribute value uses a mathematical interval notation to specify a range of bundle versions. A bundle-version attribute value specified as a single version means a version range that includes any bundle version greater than or equal to the specified version.

Since 1.3

2.23.10.24 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

2.23.10.25 public static final String EXCLUDE_DIRECTIVE = "exclude"

Manifest header directive (named "exclude") identifying a list of classes and/or resources of the specified package which must not be allowed to be exported in the Export-Package manifest header.

The directive value is encoded in the Export-Package manifest header like:

```
 Export-Package: org.osgi.framework; exclude:="MyStuff*"
```

Since 1.3

2.23.10.26 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.

2.23.10.27 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.

2.23.10.28 public static final String FRAGMENT_ATTACHMENT_ALWAYS = "always"

Manifest header attribute value (named "always") identifying a fragment attachment type of always. A fragment attachment type of always indicates that fragments are allowed to attach to the host bundle at any time (while the host is resolved or during the process of resolving the host bundle).

The attribute value is encoded in the Bundle-SymbolicName manifest header like:

```
 Bundle-SymbolicName: com.acme.module.test; fragment-attach-
ment="always"
```

See Also `Constants.FRAGMENT_ATTACHMENT_ATTRIBUTE`[p.90]

Since 1.3

2.23.10.29 public static final String FRAGMENT_ATTACHMENT_ATTRIBUTE = "fragment-attachment"

Manifest header attribute (named "fragment-attachment") identifying if and when a fragment may attach to a host bundle. The default value is "`always`".

The attribute value is encoded in the Bundle-SymbolicName manifest header like:

```
 Bundle-SymbolicName: com.acme.module.test; fragment-attach-
ment="never"
```

See Also Constants.FRAGMENT_ATTACHMENT_ALWAYS[p.90], Constants.FRAGMENT_ATTACHMENT_RESOLVETIME[p.91], Constants.FRAGMENT_ATTACHMENT_NEVER[p.91]

Since 1.3

2.23.10.30 public static final String FRAGMENT_ATTACHMENT_NEVER = "never"

Manifest header attribute value (named "never") identifying a fragment attachment type of never. A fragment attachment type of never indicates that no fragments are allowed to attach to the host bundle at any time.

The attribute value is encoded in the Bundle-SymbolicName manifest header like:

```
 Bundle-SymbolicName: com.acme.module.test; fragment-attach-
ment="never"
```

See Also Constants.FRAGMENT_ATTACHMENT_ATTRIBUTE[p.90]

Since 1.3

2.23.10.31 public static final String FRAGMENT_ATTACHMENT_RESOLVETIME = "resolve-time"

Manifest header attribute value (named "resolve-time") identifying a fragment attachment type of resolve-time. A fragment attachment type of resolve-time indicates that fragments are allowed to attach to the host bundle only during the process of resolving the host bundle.

The attribute value is encoded in the Bundle-SymbolicName manifest header like:

```
 Bundle-SymbolicName: com.acme.module.test; fragment-attach-
ment="resolve-time"
```

See Also Constants.FRAGMENT_ATTACHMENT_ATTRIBUTE[p.90]

Since 1.3

2.23.10.32 public static final String FRAGMENT_HOST = "Fragment-Host"

Manifest header (named "Fragment-Host") identifying the symbolic name of another bundle for which that the bundle is a fragment.

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

Since 1.3

2.23.10.33 public static final String FRAMEWORK_EXECUTIONENVIRONMENT = "org.osgi.framework.executionenvironment"

Framework environment property (named "org.osgi.framework.execution-environment") identifying execution environments provided by the Framework.

The value of this property may be retrieved by calling the `BundleContext.getProperty` method.

Since 1.2

2.23.10.34 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.

2.23.10.35 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.

2.23.10.36 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.

2.23.10.37 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.

2.23.10.38 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.

2.23.10.39 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.

2.23.10.40 public static final String GROUPING_DIRECTIVE = "grouping"

Manifest header directive (named "grouping") identifying the package grouping specified in the Import-Package or Export-Package manifest header.

The directive value is encoded in the Import-Package or Export-Package manifest header like:

```
Export-Package: org.osgi.framework; grouping:="coregroup"
```

Since 1.3

2.23.10.41 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.

2.23.10.42 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.

2.23.10.43 public static final String INCLUDE_DIRECTIVE = "include"

Manifest header directive (named "include") identifying a list of classes and/or resources of the specified package which must be allowed to be exported in the Export-Package manifest header.

The directive value is encoded in the Export-Package manifest header like:

```
Export-Package: org.osgi.framework; include:="MyStuff*"
```

Since 1.3

2.23.10.44 public static final String MANDATORY_DIRECTIVE = "mandatory"

Manifest header directive (named "mandatory") identifying names of matching attributes which must be specified by matching Import-Package statements in the Export-Package manifest header.

The directive value is encoded in the Export-Package manifest header like:

```
Export-Package: org.osgi.framework; mandatory:="bundle-sym-
bolic-name"
```

Since 1.3

2.23.10.45 public static final String MULTIPLE_HOSTS_DIRECTIVE = "multiple-hosts"

Manifest header directive (named "multiple-hosts") identifying if the fragment should attach to each bundle selected by the Fragment-Host manifest header. The default value is `false`.

The directive value is encoded in the Fragment-Host manifest header like:

```
Fragment-Host: com.acme.module.test; multiple-hosts=”false”
```

Since 1.3

2.23.10.46 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.

2.23.10.47 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”
```

2.23.10.48 public static final String REEXPORT_PACKAGE = “Reexport-Package”

Manifest header (named “Reexport-Package”) identifying the names of the packages that the bundle offers to the Framework for reexport.

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

Since 1.3

2.23.10.49 public static final String REQUIRE_BUNDLE = “Require-Bundle”

Manifest header (named “Require-Bundle”) identifying the symbolic names of other bundles required by the bundle.

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

Since 1.3

2.23.10.50 public static final String RESOLUTION_DIRECTIVE = “resolution”

Manifest header directive (named “resolution”) identifying the resolution type in the Import-Package or Require-Bundle manifest header.

The directive value is encoded in the Import-Package or Require-Bundle manifest header like:

```
Import-Package: org.osgi.framework; resolution:=”optional”
Require-Bundle: com.acme.module.test; resolution:=”optional”
```

See Also `Constants.RESOLUTION_MANDATORY`[p.94], `Constants.RESOLUTION_OPTIONAL`[p.95]

Since 1.3

2.23.10.51 public static final String RESOLUTION_MANDATORY = "mandatory"

Manifest header directive value (named "mandatory") identifying a mandatory resolution type. A mandatory resolution type indicates that the import package or require bundle must be resolved when the bundle is resolved. If such an import or require bundle cannot be resolved, the module fails to resolve.

The directive value is encoded in the Import-Package or Require-Bundle manifest header like:

```
 Import-Package: org.osgi.framework; resolution:="manditory"
 Require-Bundle: com.acme.module.test; resolution:="manditory"
```

See Also `Constants.RESOLUTION_DIRECTIVE`[p.94]

Since 1.3

2.23.10.52 public static final String RESOLUTION_OPTIONAL = "optional"

Manifest header directive value (named "optional") identifying an optional resolution type. An optional resolution type indicates that the import or require bundle is optional and the bundle may be resolved without the import or require bundle being resolved. If the import or require bundle is not resolved when the bundle is resolved, the import or require bundle may not be resolved before the bundle is refreshed.

The directive value is encoded in the Import-Package or Require-Bundle manifest header like:

```
 Import-Package: org.osgi.framework; resolution:="optional"
 Require-Bundle: com.acme.module.test; resolution:="optional"
```

See Also `Constants.RESOLUTION_DIRECTIVE`[p.94]

Since 1.3

2.23.10.53 public static final String SELECTION_FILTER_ATTRIBUTE = "selection-filter"

Manifest header attribute (named "selection-filter") is used for selection by filtering based upon system properties.

The attribute value is encoded in manifest headers like:

```
 Bundle-NativeCode: libgtk.so; selection-filter="(ws=gtk)";
...
 Bundle-ClassPath: base.jar, gtk.jar; selection-fil-
ter="(ws=gtk)", ...
```

Since 1.3

2.23.10.54 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.

2.23.10.55 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.

2.23.10.56 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.63]) and followed by a dot (.). A bundle may use this as the prefix of the persistent identifiers for the services it registers.

2.23.10.57 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.77] 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.

2.23.10.58 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.

2.23.10.59 public static final String SINGLETON_ATTRIBUTE = "singleton"

Manifest header attribute (named "singleton") identifying whether a bundle is a singleton. The default value is `false`.

The attribute value is encoded in the Bundle-SymbolicName manifest header like:

```
Bundle-SymbolicName: com.acme.module.test; singleton=true
```

Since 1.3

2.23.10.60 public static final String SYSTEM_BUNDLE_LOCATION = "System Bundle"

Location identifier of the OSGi *system bundle*, which is defined to be "System Bundle".

2.23.10.61 public static final String VERSION_ATTRIBUTE = "version"

Manifest header attribute (named "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; version="1.1"
```

Since 1.3

2.23.10.62 public static final String VISIBILITY_DIRECTIVE = "visibility"

Manifest header directive (named "visibility") identifying the visibility of a reqiured bundle in the Require-Bundle manifest header.

The directive value is encoded in the Require-Bundle manifest header like:

```
Require-Bundle: com.acme.module.test; visibility:="reexport"
```

See Also Constants.VISIBILITY_PRIVATE[p.97], Constants.VISIBILITY_REEXPORT[p.97]

Since 1.3

2.23.10.63 public static final String VISIBILITY_PRIVATE = "private"

Manifest header directive value (named "private") identifying a private visibility type. A private visibility type indicates that any packages that are exported by the required bundle are not made visible on the export signature of the requiring bundle.

The directive value is encoded in the Require-Bundle manifest header like:

```
Require-Bundle: com.acme.module.test; visibility:="private"
```

See Also Constants.VISIBILITY_DIRECTIVE[p.97]

Since 1.3

2.23.10.64 public static final String VISIBILITY_REEXPORT = "reexport"

Manifest header directive value (named "reexport") identifying a reexport visibility type. A reexport visibility type indicates any packages that are exported by the required bundle are re-exported by the requiring bundle. Any arbitrary arbitrary matching attributes with which they were exported by the required bundle are deleted.

The directive value is encoded in the Require-Bundle manifest header like:

```
Require-Bundle: com.acme.module.test; visibility:="reexport"
```

See Also Constants.VISIBILITY_DIRECTIVE[p.97]

Since 1.3

2.23.11 public interface Filter

An RFC 1960-based Filter.

`Filter` objects can be created by calling `BundleContext.createFilter`[p.75] 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

2.23.11.1 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.

2.23.11.2 public int hashCode( )

□ Returns the hashCode for this `Filter` object.

Returns The hashCode of the filter string; that is, `this.toString().hashCode()`.

2.23.11.3 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.

2.23.11.4 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.

2.23.11.5 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.

2.23.12 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 Alliance reserves the right to extend the set of event types.

2.23.12.1 public static final int ERROR = 2

An error has occurred.

There was an error associated with a bundle.

The value of `ERROR` is 0x00000002.

2.23.12.2 public static final int INFO = 32

An informational event has occurred.

There was an informational event associated with a bundle.

The value of `INFO` is 0x00000020.

Since 1.3

2.23.12.3 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

2.23.12.4 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`

2.23.12.5 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

2.23.12.6 public static final int WARNING = 16

A warning has occurred.

There was a warning associated with a bundle.

The value of `WARNING` is 0x00000010.

Since 1.3

2.23.12.7 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.

2.23.12.8 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.

2.23.12.9 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.

2.23.12.10 public Throwable getThrowable( )

□ Returns the exception related to this event.

Returns The related exception or `null` if none.

2.23.12.11 public int getType( )

□ Returns the type of framework event.

The type values are:

- `STARTED`[p.99]
- `ERROR`[p.99]
- `WARNING`[p.100]
- `INFO`[p.99]
- `PACKAGES_REFRESHED`[p.99]
- `STARTLEVEL_CHANGED`[p.99]

Returns The type of state change.

2.23.13 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.74] method. `FrameworkListener` objects are called with a `FrameworkEvent` objects when the Framework starts and when asynchronous errors occur.

See Also `FrameworkEvent`[p.99]

2.23.13.1 public void frameworkEvent(FrameworkEvent event)

event The `FrameworkEvent` object.

□ Receives notification of a general `FrameworkEvent` object.

2.23.14 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.98] for a description of the filter string syntax.

2.23.14.1 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.

2.23.14.2 public String getFilter( )

□ Returns the filter string that generated the `InvalidSyntaxException` object.

Returns The invalid filter string.

See Also `BundleContext.getServiceReferences`[p.78], `BundleContext.addServiceListener(ServiceListener,String)`[p.74]

2.23.15 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.

2.23.15.1 public static final String EXPORT = “export”

The action string export.

2.23.15.2 public static final String IMPORT = “import”

The action string import.

2.23.15.3 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.

2.23.15.4 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.

2.23.15.5 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.

2.23.15.6 public int hashCode( )

□ Returns the hash code value for this object.

Returns A hash code value for this object.

2.23.15.7 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.

2.23.15.8 public PermissionCollection newPermissionCollection( )

□ Returns a new `PermissionCollection` object suitable for storing `PackagePermission` objects.

Returns A new `PermissionCollection` object.

2.23.16 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 Alliance reserves the right to extend the set of types.

See Also `ServiceListener`[p.105]

2.23.16.1 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.109]

2.23.16.2 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(String[],Object,Dictionary)`[p.80]

2.23.16.3 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.109], `BundleContext.ungetService`[p.81]

2.23.16.4 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.

2.23.16.5 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.

2.23.16.6 public int getType( )

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

- `REGISTERED`[p.103]
- `MODIFIED`[p.103]
- `UNREGISTERING`[p.103]

Returns Type of service lifecycle change.

2.23.17 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.76]

2.23.17.1 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.76]

2.23.17.2 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.81]

2.23.18 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.103] , `ServicePermission`[p.106]

2.23.18.1 public void serviceChanged(ServiceEvent event)

event The `ServiceEvent` object.

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

2.23.19 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.

2.23.19.1 public static final String GET = "get"

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

2.23.19.2 public static final String REGISTER = "register"

The action string `register` (Value is "register").

2.23.19.3 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.

2.23.19.4 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.

2.23.19.5 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.

2.23.19.6 public int hashCode( )

□ Returns the hash code value for this object.

Returns Hash code value for this object.

2.23.19.7 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.

2.23.19.8 public PermissionCollection newPermissionCollection( )

□ Returns a new `PermissionCollection` object for storing `ServicePermission objects`.

Returns A new `PermissionCollection` object suitable for storing `ServicePermission` objects.

2.23.20 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.77], `BundleContext.getServiceReferences`[p.78], `BundleContext.getService`[p.76]

2.23.20.1 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(String[],Object,Dictionary)`[p.80]

2.23.20.2 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.

2.23.20.3 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(String[],Object,Dictionary)[p.80] or ServiceRegistration.setProperties[p.109] methods.

Returns An array of property keys.

2.23.20.4 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

2.23.21 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(String[],Object,Dictionary)` [p.80]

2.23.21.1 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.

2.23.21.2 public void setProperties(Dictionary properties)

properties The properties for this service. See `Constants`[p.85] 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.94] and `Constants.SERVICE_ID`[p.95] 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.103] is synchronously sent.

Throws `IllegalStateException` – If this `ServiceRegistration` object has already been unregistered.

`IllegalArgumentException` – If `properties` contains case variants of the same key name.

2.23.21.3 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.103] 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.104] 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.81], `ServiceFactory.ungetService`[p.105]

2.23.22 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.73] 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.82]

Since 1.1

2.24 References

[4] *The Standard for the Format of ARPA Internet Text Messages*
STD 11, RFC 822, UDEL, August 1982
http://www.ietf.org/rfc/rfc822.txt

[5] *The Hypertext Transfer Protocol - HTTP/1.1*
RFC 2068 DEC, MIT/LCS, UC Irvine, January 1997
http://www.ietf.org/rfc/rfc2068.txt

[6] *The Java 2 Platform API Specification*
Standard Edition, Version 1.3, Sun Microsystems
http://java.sun.com/j2se/1.4

[7] *The Java Language Specification*
Second Edition, Sun Microsystems, 2000
http://java.sun.com/docs/books/jls/index.html

[8] *A String Representation of LDAP Search Filters*
RFC 1960, UMich, 1996
http://www.ietf.org/rfc/rfc1960.txt

[9] *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

[10] *The Java 2 Package Versioning Specification*
http://java.sun.com/j2se/1.4/docs/guide/versioning/index.html

[11] *Codes for the Representation of Names of Languages*
ISO 639, International Standards Organization
http://lcweb.loc.gov/standards/iso639-2/langhome.html

[12] *Manifest Format*
http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html#JAR%20Manifest

3 Package Admin Service Specification

Version 1.1

3.1 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.

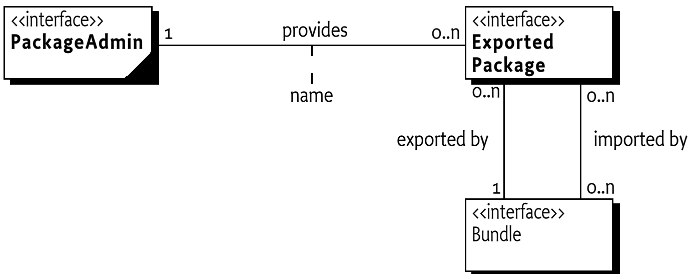
3.1.1 Essentials

- *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.

3.1.2 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 11 Class Diagram org.osgi.service.packageadmin



3.1.3 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 18. 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.

3.2 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.

3.2.1 Refreshing Packages and Start Level Service

Bundles may be stopped and later started when the refreshPackages(Bundle[]) method is called. If the Start Level Service is present, the stopping and starting of bundles must not violate the start level constraints. This implies that bundles with a higher start level must be stopped before bundles with a lower start level are stopped. Vice versa, bundles should not be started before all the bundles with a lower start level are started. See *Startup sequence* on page 127.

3.3 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.

3.4 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.

3.5 org.osgi.service.packageadmin

The OSGi Package Admin service Package. Specification Version 1.2.

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

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

3.5.1 Summary

- ExportedPackage - An exported package. [p.115]
- PackageAdmin - Framework service which allows bundle programmers to inspect the packages exported in the Framework and eagerly update or uninstall bundles. [p.116]
- RequiredBundle - A required bundle. [p.120]

3.5.2 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.

3.5.2.1 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.

3.5.2.2 public Bundle[] getImportingBundles( )

- Returns the resolved bundles that are currently importing the package associated with this `ExportedPackage` object.

Bundles which require the exporting bundle associated with this `ExportedPackage` object are considered to be importing bundles and are included in the returned array. See `RequiredBundle.getRequiringBundles()`[p.120]

Returns The array of resolved bundles currently importing the package associated with this `ExportedPackage` object, or `null` if this `ExportedPackage` object has become stale.

3.5.2.3 public String getName( )

- Returns the name of the package associated with this `ExportedPackage` object.

Returns The name of this `ExportedPackage` object.

3.5.2.4 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.

3.5.2.5 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.

3.5.3 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.115])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.119] 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.

3.5.3.1 public static final int BUNDLE_TYPE_FRAGMENT = 1

The bundle is a fragment bundle.

The value of BUNDLE_TYPE_FRAGMENT is 0x00000001.

Since 1.2

3.5.3.2 public ExportedPackage[] getAllExportedPackages(String name)

name The name of the exported packages to be returned.

□ Gets all the ExportedPackage objects 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 An array of the exported packages with the specified name, or null if no exported packages with that name exists.

Since 1.2

3.5.3.3 public Bundle getBundle(Class clazz)

clazz the class object to get a bundle for

□ Returns the bundle for which the specified class is loaded from. The classloader of the bundle returned must have been used to load the specified class. If the class was not loaded by a bundle classloader then null is returned.

Returns the bundle from which the specified class is loaded or null if the class was not loaded by a bundle classloader

Since 1.2

3.5.3.4 public Bundle[] getBundles(String symbolicName, String versionRange)

symbolicName The symbolic name of the desired bundles.

versionRange The version range of the desired bundles, or null if all versions are desired.

□ Returns the bundles with the specified symbolic name within the specified version range. If no bundles are installed that have the specified symbolic name, then null is returned. If a version range is specified, then only the bundles that have the specified symbolic name and belong to the specified version range are returned. The returned bundles are ordered by version in descending version order so that the first element of the array contains the bundle with the highest version.

Returns An array of bundles with the specified name belonging to the specified version range ordered in descending version order, or null if no bundles are found.

See Also `org.osgi.framework.Constants.BUNDLE_VERSION_ATTRIBUTE`

Since 1.2

3.5.3.5 public int getBundleType(Bundle bundle)

□ Returns the special type of the specified bundle. The bundle type values are:

- `BUNDLE_TYPE_FRAGMENT`[p.117]

If a bundle is not one or more of the defined types then 0x00000000 is returned.

Returns The special type of the bundle.

Since 1.2

3.5.3.6 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. The exported package with the highest version will be returned.

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 exported packages with that name exists.

3.5.3.7 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.

3.5.3.8 public Bundle[] getFragments(Bundle bundle)

bundle The bundle whose attached fragment bundles are to be returned.

□ Returns an array of attached fragment bundles for the specified bundle. If the specified bundle is a fragment then `null` is returned. If no fragments are attached to the specified bundle then `null` is returned.

Returns An array of fragment bundles or `null` if the bundle does not have any attached fragment bundles.

Since 1.2

3.5.3.9 public Bundle[] getHosts(Bundle bundle)

bundle The bundle whose host bundles are to be returned.

□ Returns an array of host bundles to which the specified fragment bundle is attached or `null` if the specified bundle is not attached to a host or is not a fragment bundle.

Returns An array of host bundles or `null` if the bundle does not have any host bundles.

Since 1.2

3.5.3.10 public RequiredBundle[] getRequiredBundles(String symbolicName)

symbolicName The symbolic name of the RequiredBundle or `null` for all RequiredBundles in the Framework.

□ Returns an array of RequiredBundles with the specified symbolic name. If the symbolic name argument is `null` then all RequiredBundles are returned.

Returns An array of RequiredBundles with the specified symbolic name or `null` if no RequiredBundles exist with that symbolic name.

Since 1.2

3.5.3.11 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.

3.5.3.12 public boolean resolveBundles(Bundle[] bundles)

bundles The bundles to resolve or `null` to resolve all unresolved bundles installed in the Framework.

☐ Resolve the specified bundles. The Framework must attempt to resolve the specified bundles that are unresolved. Additional bundles that are not included in the specified bundles may be resolved as a result of calling this method. A permissible implementation of this method is to attempt to resolve all unresolved bundles installed in the framework.

If `null` is specified then the Framework will attempt to resolve all unresolved bundles. This method must not cause any bundle to be refreshed, stopped, or started. This method will not return until the operation has completed.

Returns `true` if all specified bundles are resolved;

Since 1.2

3.5.4 public interface RequiredBundle

A required bundle. Instances implementing this interface are created by the Package Admin service.

The information about a `RequiredBundle` provided by this object is valid only until the next time `PackageAdmin.refreshPackages()` called. If a `RequiredBundle` object becomes stale (that is, the bundle it references has been updated or removed as a result of calling `PackageAdmin.refreshPackages()`), its `getSymbolicName()` and `getVersion()` continue to return their old values, `isRemovalPending()` returns true, and `getBundle()` and `getRequiringBundles()` return `null`.

Since 1.2

3.5.4.1 public Bundle getBundle( )

☐ Returns the bundle which defines this RequiredBundle.

Returns The bundle, or `null` if this `RequiredBundle` object has become stale.

3.5.4.2 public Bundle[] getRequiringBundles( )

- ☐ Returns the resolved bundles that currently require this bundle. If this `RequiredBundle` object is required and re-exported by another bundle then all the requiring bundles of the re-exporting bundle are included in the returned array.

Returns An array of resolved bundles currently requiring this bundle, or `null` if this `RequiredBundle` object has become stale.

3.5.4.3 public String getSymbolicName( )

- ☐ Returns the symbolic name of the bundle.

Returns The symbolic name of the bundle.

3.5.4.4 public String getVersion( )

- ☐ Returns the version of the bundle.

Returns The version of the bundle.

3.5.4.5 public boolean isRemovalPending( )

- ☐ Returns `true` if the bundle has been updated or uninstalled.

Returns `true` if the bundle has been updated or uninstalled, or if the `RequiredBundle` object has become stale; `false` otherwise.

4 Start Level Service Specification

Version 1.0

4.1 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.

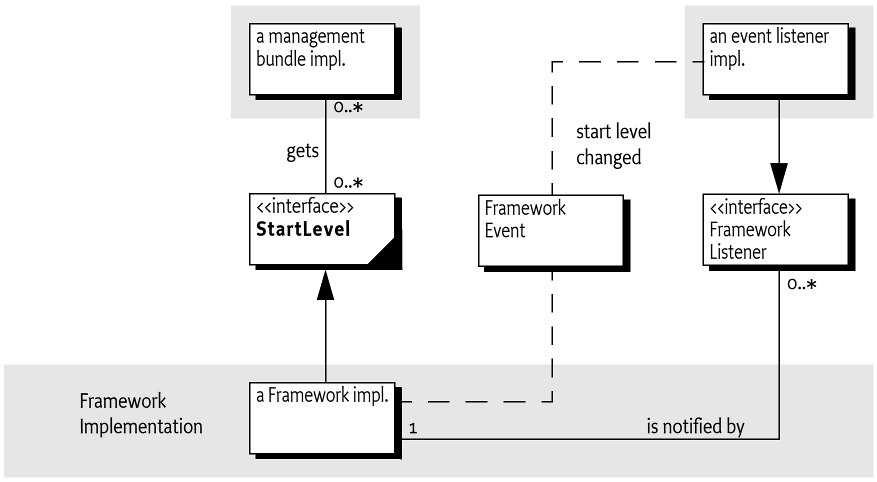
4.1.1 Essentials

- *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.

4.1.2 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 99.
- *Framework Listener* – See page 100.

Figure 12 *Class Diagram org.osgi.service.startlevel package*



4.2 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.

4.2.1 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.

4.2.2 Changing the Active Start Level

A Management Agent can influence the active start level with the set-StartLevel(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 set-StartLevel(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.

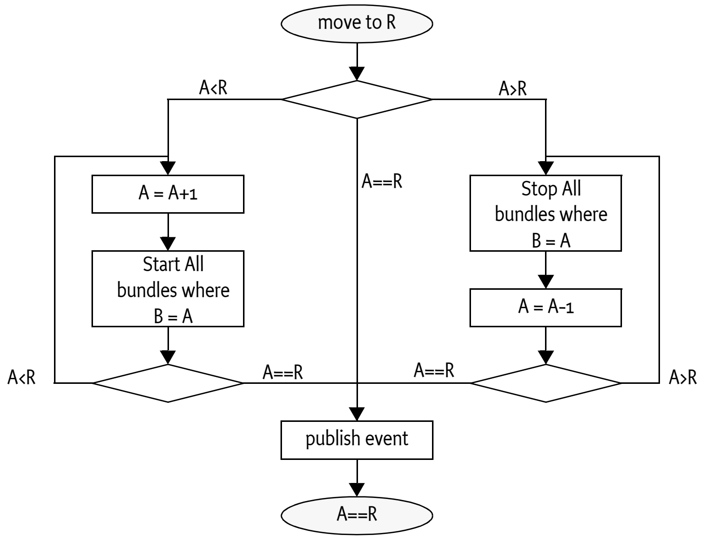
When the requested start level is reached and all bundles satisfy the condition that their bundle start level <= active start level in order to be started, then the FrameworkEvent.STARTLEVEL_CHANGED event must be sent to all registered FrameworkListener objects. If the requested start level and active 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 13.

Figure 13 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.

4.2.3 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 125 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.

4.2.4 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 125 to make the active start level equal to zero.

4.2.5 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.

4.2.6 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.

4.2.7 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 or the bundle's start level, then the Exception must be wrapped in a BundleException and broadcast as a FrameworkEvent.ERROR event. Otherwise a new BundleException must be created containing the exception and this BundleException is then thrown to the caller.

4.2.8 System Bundle

The System Bundle is defined to have a start level of zero. See page 14 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.

4.3 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.

4.4 Example Applications

The Start Level service allows a Management Agent to implement many different startup schemes. The following sections show some examples.

4.4.1 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.

4.4.2 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() { ... }
}
```

4.5 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].

4.6 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
```

4.6.1 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.

4.6.1.1 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.

4.6.1.2 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.131]

4.6.1.3 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.

4.6.1.4 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.

4.6.1.5 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.

4.6.1.6 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.

4.6.1.7 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.

5 Permission Admin Service Specification

Version 1.1

5.1 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 53.

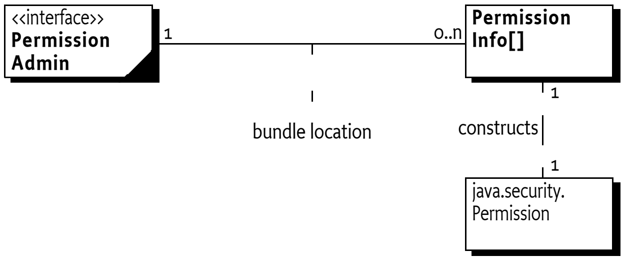
5.1.1 Essentials

- *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.

5.1.2 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 29.

Figure 14   Class Diagram org.osgi.service.permissionadmin.



5.1.3 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.

5.2 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.

5.2.1 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.

5.3 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.

5.4 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.

5.5 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; specifica-
tion-version=1.1
```

5.5.1 Summary

- PermissionAdmin - The Permission Admin service allows management agents to manage the permissions of bundles. [p.138]
- PermissionInfo - Permission representation used by the Permission Admin service. [p.140]

5.5.2 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`.

5.5.2.1 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.

5.5.2.2 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.

5.5.2.3 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.

5.5.2.4 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`.

5.5.2.5 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`.

5.5.3 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.

5.5.3.1 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`.

5.5.3.2 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.141]

5.5.3.3 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.

5.5.3.4 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.

5.5.3.5 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 "<i>name</i>")
```

or

```
  (type "<i>name</i>" "<i>actions</i>")
```

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`.

5.5.3.6 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.

5.5.3.7 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`.

5.5.3.8 public int hashCode( )

- Returns the hash code value for this object.

Returns A hash code value for this object.

5.5.3.9 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`.

6 Conditional Permission Admin Specification

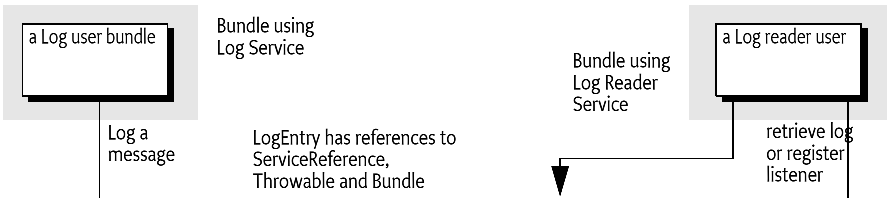
Version 1.0

6.1 Introduction

6.1.1 Entities

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

Figure 15 Log Service Class Diagram org.osgi.service.log package



6.2 The Conditional Permission Admin Service

6.3 Security

6.4 org.osgi.service.condpermadmin

The OSGi Conditional Permission Admin Specification Version 1.0.

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

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

6.4.1 Summary

- BundleLocationCondition - [p.144]
- Condition - [p.144]

- ConditionalPermissionAdmin - [p.144]
- ConditionalPermissionInfo - [p.144]
- ConditionInfo - [p.144]

6.4.2 public class BundleLocationCondition implements Condition

6.4.2.1 public BundleLocationCondition(Bundle b, String location)

6.4.2.2 public boolean isEvaluated( )

6.4.2.3 public boolean isMutable( )

6.4.2.4 public boolean isSatisfied( )

6.4.2.5 public boolean isSatisfied(Condition[] conds, Dictionary oneShotData)

6.4.3 public interface Condition

6.4.3.1 public boolean isEvaluated( )

6.4.3.2 public boolean isMutable( )

6.4.3.3 public boolean isSatisfied( )

6.4.3.4 public boolean isSatisfied(Condition[] conds, Dictionary context)

6.4.4 public interface ConditionalPermissionAdmin

6.4.4.1 public ConditionalPermissionInfo addCollection(ConditionInfo[] conds, PermissionInfo[] perms)

6.4.4.2 public Enumeration getCollections( )

6.4.5 public interface ConditionalPermissionInfo

6.4.5.1 public void delete( )

6.4.5.2 public ConditionInfo[] getConditionInfos( )

6.4.5.3 public PermissionInfo[] getPermissionInfos( )

6.4.6 public final class ConditionInfo

6.4.6.1 public ConditionInfo(String encodedCondition)

6.4.6.2 public ConditionInfo(String type, String[] args)

6.4.6.3 public boolean equals(Object obj)

6.4.6.4 public String[] getArgs( )

6.4.6.5 public String getEncoded( )

6.4.6.6 public String getType( )

6.4.6.7 public int hashCode( )

6.4.6.8 public String toString( )

7 URL Handlers Service Specification

Version 1.0

7.1 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 1.

7.1.1 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.

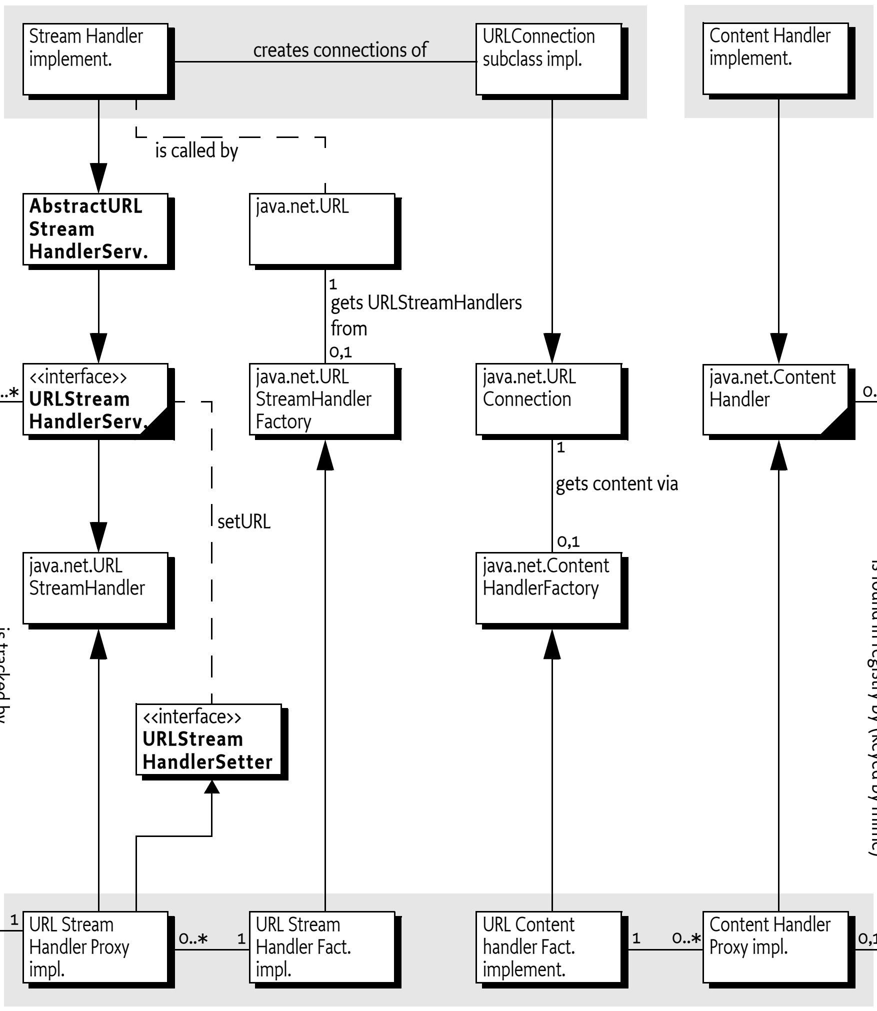
7.1.2 Entities

- *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 [15] *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 16 *Class Diagram, java.net (URL and associated classes)*


Stream Handler implement.
creates connections of
URLConnection subclass impl.
Content Handler implement.
is called by
AbstractURL Stream HandlerServ.
java.net.URL
1
gets URLStreamHandlers from
0,1
<<interface>> URLStream HandlerServ.
0..*
java.net.URL StreamHandler Factory
java.net.URL Connection
java.net.Content Handler
0..*
1
gets content via
0,1
setURL
java.net.URL StreamHandler
java.net.Content HandlerFactory
is tracked by
is found in registry by (keyed by mime)
<<interface>> URLStream HandlerSetter
1
URL Stream Handler Proxy impl.
0..*
1
URL Stream Handler Fact. impl.
URL Content handler Fact. implement.
1
0..*
Content Handler Proxy impl.
0,1

7.1.3 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.

7.2 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 [13] *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.

7.3 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 [15] *MIME Multipurpose Internet Mail Extension.*

7.3.1 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 41) 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 handle the error.

In the case of a URL Stream Handler proxy, it 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.

In the case of a Content Handler proxy, it must return InputStream to the data.

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.

7.3.2 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 must never be overridden.

Need a reference here to the EE

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.

7.3.3 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
```

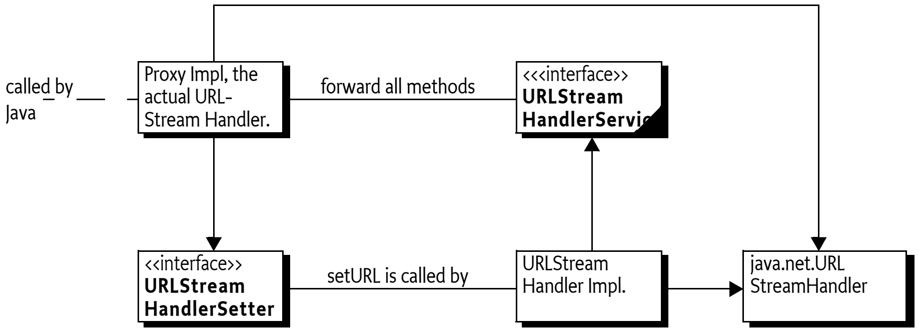
The Java run-time specific packages should be listed in the appropriate properties so that implementations of the URL Stream Handler Factory and Content Handler Factory can be made aware of these packages.

7.3.4 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 methods 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 17 *Proxy Issues*


called by Java
Proxy Impl, the actual URL-Stream Handler.
forward all methods
<<<interface>> URLStream HandlerServic
<<interface>> URLStream HandlerSetter
setURL is called by
URLStream Handler Impl.
java.net.URL StreamHandler

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.

7.4 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";
    }
 }
```

7.5 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()) >= 0 )
         sb.append( (char) c );
      r.close(); in.close();
      return sb.toString();
   }
}
```

7.6 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.

7.7 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-ver-
sion=1.0
```

7.7.1 Summary

- AbstractURLStreamHandlerService - Abstract implementation of the URLStreamHandlerService interface. [p.155]
- URLConstants - Defines standard names for property keys associated with URLStreamHandlerService[p.157] and java.net.ContentHandler services. [p.157]
- URLStreamHandlerService - Service interface with public versions of the protected java.net.URLStreamHandler methods. [p.157]
- URLStreamHandlerSetter - Interface used by URLStreamHandlerService objects to call the setURL method on the proxy URLStreamHandler object. [p.158]

7.7.2 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.

7.7.2.1 protected URLStreamHandlerSetter realHandler

The URLStreamHandlerSetter object passed to the parseURL method.

7.7.2.2 public AbstractURLStreamHandlerService( )

7.7.2.3 public boolean equals(URL u1, URL u2)

□ This method calls super.equals(URL,URL).

See Also java.net.URLStreamHandler.equals(URL,URL)

7.7.2.4 **public int getDefaultPort( )**

- This method calls `super.getDefaultPort`.

See Also `java.net.URLStreamHandler.getDefaultPort`

7.7.2.5 **public InetAddress getHostAddress(URL u)**

- This method calls `super.getHostAddress`.

See Also `java.net.URLStreamHandler.getHostAddress`

7.7.2.6 **public int hashCode(URL u)**

- This method calls `super.hashCode(URL)`.

See Also `java.net.URLStreamHandler.hashCode(URL)`

7.7.2.7 **public boolean hostsEqual(URL u1, URL u2)**

- This method calls `super.hostsEqual`.

See Also `java.net.URLStreamHandler.hostsEqual`

7.7.2.8 **public abstract URLConnection openConnection(URL u) throws IOException**

See Also `java.net.URLStreamHandler.openConnection`

7.7.2.9 **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`

7.7.2.10 **public boolean sameFile(URL u1, URL u2)**

- This method calls `super.sameFile`.

See Also `java.net.URLStreamHandler.sameFile`

7.7.2.11 **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.

7.7.2.12 **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)`

7.7.2.13 public String toExternalForm(URL u)

- This method calls `super.toExternalForm`.

See Also `java.net.URLStreamHandler.toExternalForm`

7.7.3 public interface URLConstants

Defines standard names for property keys associated with `URLStreamHandlerService`[p.157] and `java.net.ContentHandler` services.

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

7.7.3.1 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.

7.7.3.2 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.

7.7.4 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.158] 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.155]

7.7.4.1 public boolean equals(URL u1, URL u2)

See Also `java.net.URLStreamHandler.equals(URL, URL)`

7.7.4.2 public int getDefaultPort( )

See Also `java.net.URLStreamHandler.getDefaultPort`

7.7.4.3 public InetAddress getHostAddress(URL u)

See Also `java.net.URLStreamHandler.getHostAddress`

7.7.4.4 public int hashCode(URL u)

See Also `java.net.URLStreamHandler.hashCode(URL)`

7.7.4.5 public boolean hostsEqual(URL u1, URL u2)

See Also `java.net.URLStreamHandler.hostsEqual`

7.7.4.6 public URLConnection openConnection(URL u) throws IOException

See Also `java.net.URLStreamHandler.openConnection`

7.7.4.7 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`

7.7.4.8 public boolean sameFile(URL u1, URL u2)

See Also `java.net.URLStreamHandler.sameFile`

7.7.4.9 public String toExternalForm(URL u)

See Also `java.net.URLStreamHandler.toExternalForm`

7.7.5 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.158] 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.

7.7.5.1 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.

7.7.5.2 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)`

7.8 References

[13] *Java .net*
http://java.sun.com/j2se/1.4/docs/api/java/net/package-summary.html

[14] *URLs*
http://www.ietf.org/rfc/rfc1738.txt

[15] *MIME Multipurpose Internet Mail Extension*
http://www.nacs.uci.edu/indiv/ehood/MIME/MIME.html

[16] *Assigned MIME Media Types*
http://www.iana.org/assignments/media-types

8 Service Tracker Specification

Version 1.2

8.1 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.

8.1.1 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.

8.1.2 Operation

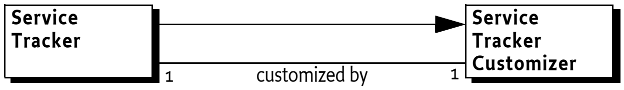
The fundamental tasks of a `ServiceTracker` object are:

- 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.

8.1.3 Entities

Figure 18 *Class diagram of org.osgi.util.tracker*



8.1.4 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.

8.2 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.

8.3 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.

8.4 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.

8.4.1 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.

8.5 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);
}
```

8.6 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.

8.7 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.

8.8 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-ver-
sion=1.2
```

8.8.1 Summary

- ServiceTracker - The `ServiceTracker` class simplifies using services from the Framework's service registry. [p.161]
- ServiceTrackerCustomizer - The `ServiceTrackerCustomizer` interface allows a `ServiceTracker` object to customize the service objects that are tracked. [p.161]

8.8.2 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.

8.8.2.1 protected final BundleContext context

Bundle context this `ServiceTracker` object is tracking against.

8.8.2.2 protected final Filter filter

Filter specifying search criteria for the services to track.

Since 1.1

8.8.2.3 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.

8.8.2.4 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.

8.8.2.5 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

8.8.2.6 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.161]

8.8.2.7 public synchronized void close( )

□ Close this `ServiceTracker` object.

This method should be called when this `ServiceTracker` object should end the tracking of services.

8.8.2.8 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.

8.8.2.9 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.

8.8.2.10 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

8.8.2.11 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.

8.8.2.12 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.

8.8.2.13 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

8.8.2.14 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.161]

8.8.2.15 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.

8.8.2.16 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.

8.8.2.17 public void removedService(ServiceReference reference, Object service)

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.161]

8.8.2.18 public int size( )

□ Return the number of services being tracked by this `ServiceTracker` object.

Returns Number of services being tracked.

8.8.2.19 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.

8.8.3 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 `ServiceEvent` s 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.

8.8.3.1 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.

8.8.3.2 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.

8.8.3.3 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.

End Of Document