OSGi Service Platform, Core Specification Release 4

Draft March 9, 2005 9:17 am

OSGi Service Platform Core Specification

The OSGi Alliance

Release 4
August 2005

IOS Press
Ohmsha

Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

Table Of Contents

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

Publisher

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

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

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

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

Distributor in Japan

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

Trademarks

Feedback

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

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

OSGi Member Companies

4DHomeNet, Inc.	Acunia
Alpine Electronics Europe Gmbh	AMI-C
Atinav Inc.	BellSouth Telecommunications, Inc.
BMW	Bombardier Transportation
Cablevision Systems	Coactive Networks

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 OSGi Framework Overview

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 applications known as *bundles*.

OSGi-compliant devices can download and install OSGi bundles, and remove them when they are no longer required. 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.

The functionality of the Framework is divided in the following layers:

- Security Layer (optional)
- Module Layer
- Life Cycle layer
- Service Layer
- Actual Services

The security layer is based on Java 2 security but adds a number of constraints and fills in some of the blanks that standard Java leaves open. The security layer is described in *Security Layer* on page 11.

The Module layer defines a modularization model for Java. It addresses some of the shortcomings of Java's deployment model. The modularization layer has strict rules defining how Java packages can be shared between bundles or hidden from other bundles. The module layer can be used without the life cycle and service layer. The module layer is described in *Module Layer* on page 29.

The Life Cycle layer provides a life cycle API to bundles. This API provides a runtime model for bundles. It is defined how bundles are started and stopped as well as how bundles are installed, updated and uninstalled. Additionally, it provides a comprehensive event API to allow a management bundle to control the operations of the service platform. The Life Cycle layer requires the module layer but the security layer is optional. This layer can be found at *Life Cycle Layer* on page 81

The service layer provides provides a dynamic, concise and consistent programming model for Java bundle developers, simplifying the development and deployment of service bundles 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 in many different dimensions – 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.

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 service layer is described in *Service Layer* on page 161.

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.

1.2 Sections

1.3 What is New

1.4 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.

1.5 Conventions and Terms

1.5.1 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.

1.5.2 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”.

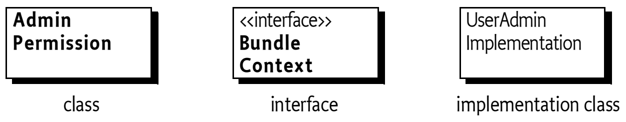
Service objects are registered with the OSGi Framework. Registration consists of the service object, a set of properties, and a list of classes and interfaces implemented by this service object. The classes and interfaces are used for type safety *and* naming. Therefore, it is said that a service object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

1.5.3 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.

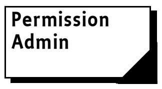
Classes or interfaces are depicted as rectangles, as in Figure 1. Interfaces are indicated with the qualifier `<<interface>>` as the first line. The name of the class/interface is indicated in bold when it is part of the specification. Implementation classes are sometimes shown to demonstrate a possible 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



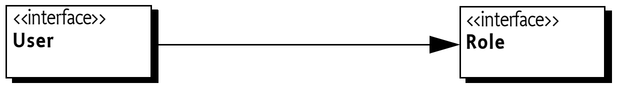
If an interface or class is used as a service object, it will have a black triangle in the bottom right corner.

Figure 2 Service symbol



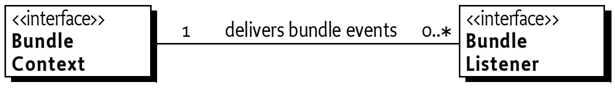
Inheritance (the `extends` or `implements` keyword in Java class definitions) is indicated with an arrow. Figure 3 shows that `User` implements or extends Role.

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."

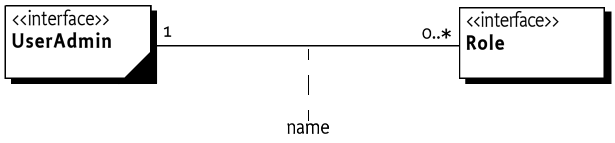
Figure 4 *Relations symbol*



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.

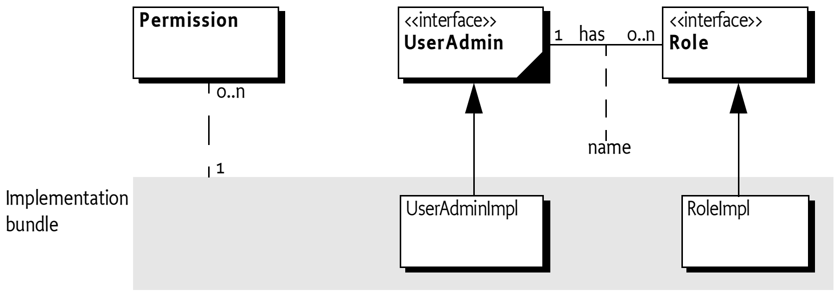
When a relationship is qualified by a name or an object, it is indicated by drawing a dotted line perpendicular to the relation and connecting this line to a class box or a description. Figure 5 shows that the relationship between a UserAdmin class and a Role class is qualified by a name. Such an association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



1.5.4 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.

1.6 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).

1.7 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
Package Admin service	org.osgi.service.packageadmin	1.1
Conditional Permission Admin Service	org.osgi.service.condpermadmin	1.0
Bundle Start Levels	org.osgi.service.startlevel	1.0
URL Stream and Content	org.osgi.service.url	1.0
Service Tracker	org.osgi.util.tracker	1.2

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.

1.8 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.

1.8.1 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.

1.9 References

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

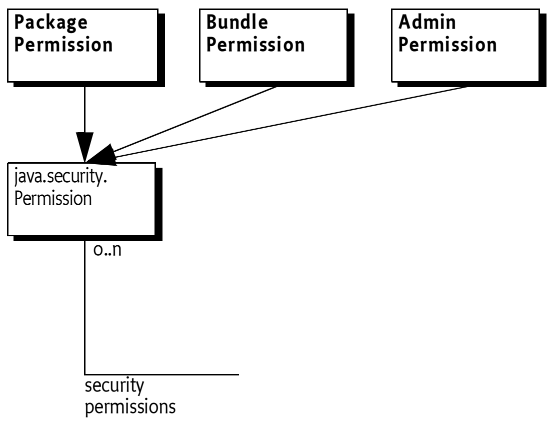
[2] *OSGi Service Gateway Specification 1.0* http://www.osgi.org/resources/spec_download.asp

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

2 Security Layer

### intro

Figure 7 *Class Diagram* org.osgi.framework



2.1 Security Architecture

The Framework security model is based on the Java 2 specification. If security checks are performed, they must be done according to [8] *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.

2.1.1 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 [8] *The Java Security Architecture for JDK 1.2*.

2.1.2 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.

2.1.3 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.

2.1.4 BundlePermission

### work

2.1.5 Resource permissions

In order to have access to a resource in a bundle, appropriate permissions are required. A bundle must always be given permission by the Framework to access the resources contained within itself, as well as permission to access any resources in imported packages.

When a URL object to a resource within a bundle is created, the caller is checked for Admin Permission to access the resource if the resource is located in another bundle. If the caller does not have the necessary permission, the resource is not accessible and Security Exception is thrown. If the caller has the necessary permission, then the URL object to the resource is successfully created. Once the URL object is created, no further permission checks are performed when the contents of the resource are accessed.

2.1.6 URLs for Resources

Why don't we standardize the scheme??

A resource in a bundle can be accessed through the class loader of that bundle but it can also be accessed with the getResource or getEntry methods. All these methods return a URL object. The URLs are called *bundle resource urls*. The schemes for getResource and getEntry can differ and are implementation dependent.

Bundle resource urls are normally created by the Framework, however, in certain cases bundles need to manipulate the URL to find related resources. The Framework is therefore required to ensure that:

- Bundle resource urls must be hierarchical (See RFC 2396: Uniform Resource Identifiers URI: Generic Syntax [6])
- Able to be used as a context for constructing another URL.
- The java.net.URLStreamHandler class used for a bundle resource URL must be available to the java.net.URL class to setup a URL that uses the protocol scheme defined by the Framework.

### How do you find this if the scheme is not defined?

- The getPath method for a bundle resource URL must return an absolute path (a path that starts with '/') to a resource or entry in a bundle. For example, the URL returned from getEntry("myimages/test.gif") must have a path of /myimages/test.gif.

For example, a class can take a URL to an index.html bundle resource and map urls in this resource to other files in the same JAR director.

```
public class BundleResource implements HttpContext {
   URL     root;  // to index.html in bundle
   URL getResource( String resource ) {
      return new URL( root, resource );
   }
   ...
}
```

#### test

URLs that are not constructed by the Framework, i.e. from bundles, must follow slightly different security rules due to the design of the java.net.URL class. Unfortunately, not all constructors of the URL class interact with the URL Stream Handler classes (the implementation specific part). Other constructors call at least the parseURL method in the URL Stream Handler where the security check can take place. This design makes it impossible for the Framework to *always* check the permissions during construction.

The following constructors use the parseURL method and are therefore checked when the URL is constructed:

```
URL(String spec)
URL(URL context, String spec)
URL(URL context, String spec, URLStreamHandler handler)
```

When one of these constructors is called for a URL, the implementation of the Framework must check in the `parseURL` method the caller for the necessary permissions. If the caller does not have the necessary permissions then a Security Exception must be thrown. If the caller has the necessary permissions, then the URL object is setup to access the bundle resource without further checks.

The following java.net.URL constructors do not call `parseURL` to setup a URL to use a specific protocol; hence making it impossible for the Framework to verify the permission during construction:

```
URL(String protocol, String host, int port, String file)
URL(String protocol, String host, int port, String file,
URLStreamHandler handler)
URL(String protocol, String host, String file)
```

Bundle resource urls that are constructed with these constructors can not perform the permission check during creation and must therefore delay the permission check. When the URL is accessed, the Framework must throw a Security Exception if the callers do not have Admin Permission when the resource is from another bundle.

2.2 Digitally Signed JAR Files

Digitally signing is a security feature that verifies the following:

- Authenticates the signer, the so called *principal*.
- Ensures that the content hasn't been modified after it was signed by the principal.

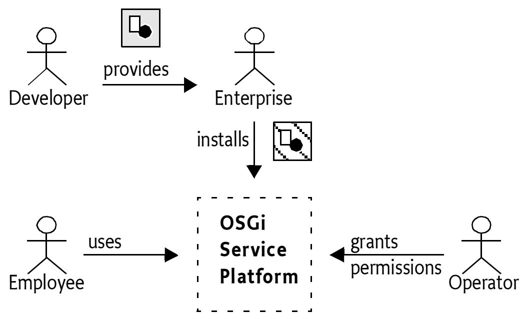
In an OSGi Framework, the principals that signed a JAR become associated with that JAR. This association is then used to:

- Grant permissions to a JAR based on the authenticated principal
- Target a set of bundles by principal for a permission to operate on or with those bundles

For example, an Operator can grant the ACME company the right to use networking on their devices. The ACME company can then use networking in every bundle they digitally sign and deploy on the Operator's device. Also, a specific bundle can get permission to only manage the life cycle of bundles that are signed by the ACME company.

Signing provides a powerful delegation model. It allows an Operator to grant a restricted set of permissions to a company, after which the company can create JARs that can use those permissions, without requiring any intervention of, or communication with, the Operator for each particular JAR. This delegation model is shown graphically in Figure 8.

Figure 8 *Delegation model*



Digitally signing is based on *public key cryptography*. Public key cryptography uses a system where there are two mathematically related keys: a public and a private key. The public key is shared with the world and can be dispersed freely, usually in the form of a certificate. The private key must be kept a secret.

Messages encrypted with a public key can only be read with the related private key. This makes it possible to send a message to a person and be sure only that person can read it. Messages encrypted with the private key can only be decrypted with the public key. This can be used to send a message to a person that can then trust the sender (assuming the public key is trusted, this is discussed in *Certificates* on page 20). This authentication mechanism is used in digital signing.

The digital signing process used is based on Java 2 Jar signing, as defined in [6] *Java Jar Signing*. The process of signing is repeated, restricted and augmented here to improve the inter-operability of OSGi bundles.

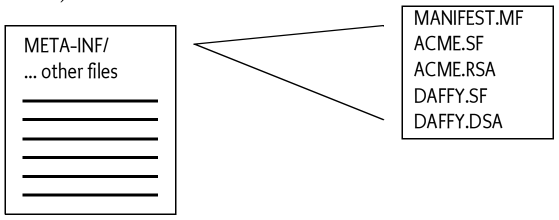
2.2.1 JAR Structure and Manifest

A JAR can be signed by multiple signers. Each signer must store two resources in the JAR file:

- A signature instruction resource that is that a similar format like the Manifest. It must have a .SF extension. It provides digests for the manifest Name section for each resource in the JAR file.
- A PKCS#7 resource that contains the digital signature of the signature instruction resource. See [17] *Public Key Crypography Standard #7* for information about its format.

These JAR file signing resources must be placed in the META-INF directory. These resources must come directly after the MANIFEST.MF file, and before *any* other resources in the JAR stream. This ordering is important because it allows the receiver of the JAR file to stream the contents without buffering. All the security information is available before any resources are loaded. This model is shown in Figure 9.

Figure 9 *Signer files in JAR*



The signature instruction resource contains digests of the appropriate information in the Manifest resource, not the actual resource data itself. A digest is a one way function that computes a value from the bytes of a resource in such a way that it is very difficult to create a set of byte that matches that value.

The JAR manifest must therefore contain one ore more digests of the actual resources. These digests must be placed in their name section of the manifest. The name of the digest header is constructed with its algorithm followed by `-Digest`. For example `SHA1-Digest`. OSGi Framework implementations must support the following digest algorithms.

- *MD5* – Message Digest 5, an improved version of MD4. It generates a 128-bit hash. It is described at page 436 in [12] *Secure Hash Algorithm 1*.
- *SHA1* – An improved version of SHA, delivers a 160 bit hash. It is defined in [13] *RFC 1321 The MD5 Message-Digest Algorithm*.

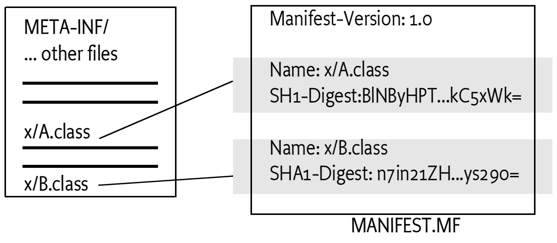
The hash must be encoded with a Base 64 encoding. Base 64 encoding is defined in [14] *RFC 1421 Privacy Enhancement for Internet Electronic Mail*.

For example, a manifest could look like:

```
Manifest-Version: 1.0
Bundle-Name: DisplayManifest
↵
Name: x/A.class
SHA1-Digest: RJpDp+igoJ1kxs8CSFeDtMbMq78=
↵
Name: x/B.class
SHA1-Digest: 3EuIPcx414w2QfFSXSZEBfLgKYA=
↵
```

Graphically this looks like Figure 10.

Figure 10 *Signer files in JAR*



OSGi JARs can be signed by multiple signers. However, and this is different from standard Java JAR signing, each signer must sign all resources except for resources that reside directly in the META-INF directory. I.e. there is no partial signing for an OSGi JAR. This specification only supports fully signed bundles. The reason for this restriction is because it partially signing can break the protection of private packages that is supported by the OSGi Framework module layer.

Signature files in nested JAR files (JARs on the Bundle-ClassPath) must be ignored. These nested JAR files must share the same protection domain as their containing bundle and not treated any differently as resources that are stored directly in the JAR.

Each signature is based on two files. The first file is the signature instruction file, this file has a file name with an extension .SF. A signature file has the same syntax as the manifest, except that it starts with Signature-Version: 1.0 instead of Manifest-Version: 1.0.

Its name sections contain digests of their corresponding name sections in the manifest. I.e. the signature instruction resource's entries digest the manifest entries and not the resource itself. This indirection is thought to improve the performance. Once the manifest and signature files are loaded, all security verification can take place. Later when resources are loaded, the loader only has to verify the resource's digest against the manifest digests.

The digest in the signature instruction resource must contain the whole name section for that file. This starts at the 'N' of the Name header, and must include the carriage return, line feed part that separates it from the next section.

For example, the section:

```
Name: OSGI-OPT/src/org/osgi/service/jini/JiniDriver.java
SHA1-Digest: RJpDp+igoJ1kxs8CSFeDtMbMq78=
MD5-Digest: HJuJUsmas67as9+ytATs
↵
```

Must look in the signature file like:

```
Name: OSGI-OPT/src/org/osgi/service/jini/JiniDriver.java
SHA1-Digest: IIsI6HranRNHMY27SK8M5qMunR4=
```

### This is not true but I cannot find it in the specification. Ben can you help me here? I tried it out, but my answer is always different

String s = "Name: OSGI-OPT/src/org/osgi/service/jini/JiniDriver.java\r\nSHA1-Digest: RJpDp+igoJ1kxs8CSFeDtMbMq78=\r\n";

java.security.MessageDigest md = java.security.MessageDigest.getInstance("SHA1");

byte [] bytes = s.getBytes("UTF-8");

md.update(bytes);

byte [] hash = md.digest();

System.out.println(scrap.Base64.byteArrayToBase64(hash));

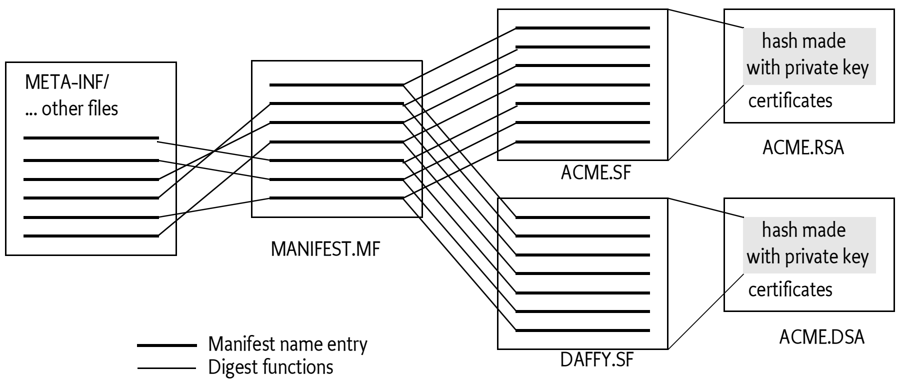
The main section in the manifest is digested in the main section of the signature file. The header name is composed of the digest algorithm (MD5 or SHA1) followed by -Digest-Manifest. The digest must be a digest of the complete manifest file.

### This is currently best practice in the JAR file but not mentioned by the the JAR spec ... However, in 1.5 they suddenly talk about: MD5-Digest-Manifest-Main-Attributes.???? What shall we take??

For a valid OSGi JAR file, the signature instruction resource must contain an entry in its name section for each resource. If there are multiple signers, then their signature instruction resources can be identical. However, each signer must still have its own signature instruction file. I.e. it is not allowed to share this file.

The indirection of the signature instruction files digests is depicted in Figure 11 for two signers: ACME and DAFFY.

Figure 11 *Manifest, signature instruction files and digests in JAR*



2.2.2 Signing Algorithms

There are different algorithms available that can perform digital signing. An OSGi Framework must support the following algorithms:

- *DSA* – The Digital Signature Algorithm. This standard is defined in [15] *DSA*. This is a USA government standard for Digital Signature Standard. The signature resource name must have an extension of .DSA.
- *RSA* – Rivest, Shamir and Adleman. A public key algorithm that is very popular. It is defined in [16] *RSA*. The extension of the signature resource name must be .RSA.

The signature files for RSA and DSA are stored in a PCKS#7 format. This is a format that has a structure as defined in [17] *Public Key Crypography Standard #7*. The PKCS#7 standard provides access to the algorithm specific signing information as well as the certificate with the public key of the signer. The verification algorithm uses the public key to verify that:

- The digital signature matches the signature instruction resource
- The signature was created with the private key associated with the certificate.

The complete signing structure is shown in Figure 11.

2.2.3 Certificates

A certificate is a general term for a signed document containing a name and public key information. Such a certificate can take many forms but the OSGi JAR signing is based on the X.509 certificate format. It has been around for many years and is part of the OSI group of standards. X.509 is defined in [5] *X.509 Certificates*.

An X.509 certificate contains the following elements:

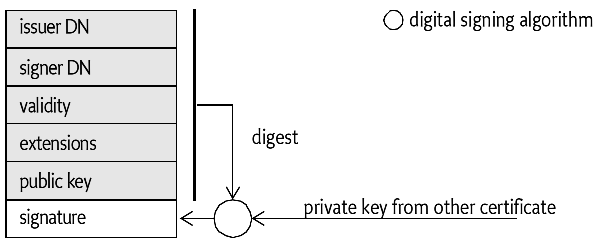
- *Subject Name* – A unique identifier for the object being certified. In the case of a person this might include the name, nationality and email address, the organization, and the department within that organization. This identifier is a Distinguished Name, which is defined in *Distinguished Names* on page 21.
- *Issuer Name* – The Distinguished Name for the principal that signed this certificate.
- *Certificate Extensions* – A certificate can also include pictures, codification of fingerprints, passport number, and so on.
- *Public Key Information* – A public key can be uses an encryption technique that requires its private counterpart to decrypt, and vice versa. The public key can be shared freely, the private key must be kept secret. The public key must be a DSA key or an RSA key. RSA is more popular.

### I made the DSA up, is that true?

- *Validity* – A Certificate can be valid for only a limited time.
- *Certifying Authority Signature* – The Certificate Authority signs the first elements and thereby adds credibility to the certificate. The receiver of a certificate can check the signature against a set of trusted certifying authorities. If the receiver trusts that certifying authority, it can trust the statement that the certificate makes.

The structure of a certificate is depicted in Figure 12.

Figure 12 *Structure of a certificate*

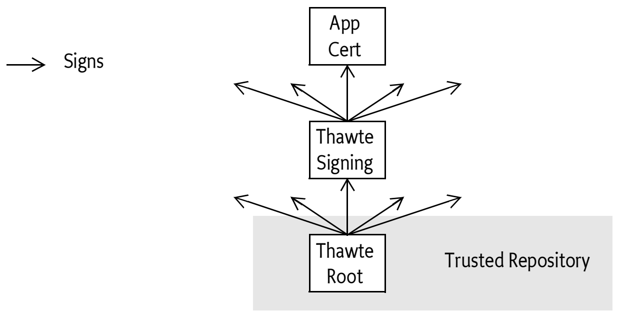


Certificates can be freely dispersed, they do not contain any secret information. Therefore, the PKCS#7 resource contains the signing certificate. It can not be trusted at face value because the certificate is carried in the bundle itself. I.e. a perpetrator can easily create its own certificate with any content. The receiver can only verify that the certificate was created by the owner of the public key and that it has not been tampered with. However, before the statement in the certificate can be trusted, it is necessary to authenticate the certificate itself. It is therefore necessary to establish a *trust model.*

One trust model, which is implied in the OSGi specifications but not specified, is to place the signing certificate in a repository in the Framework. Any certificate in this repository is treated as trusted by default. However, placing all possible certificates in this repository does not scale well. In an open model, a device would have to contain hundreds of thousands of certificates. The management of the certificates could easily become overwhelming.

The solution is to sign a certificate by another certificate, and this process can be repeated several times. This delegation process forms a *chain of certificates*. All certificates for this chain are carried in the PKCS#7 file, if one of those certificates can be found in the trusted repository, the other ones can be trusted, under the condition that all certificates are valid. This model scales very well because only a few certificates of trusted signers need to be maintained. This is the model that is used in for example web browsers.This is depicted in Figure 13.

Figure 13 *Certificate authorities fan out*



This specification does not specify access to the trusted repository. It is implementation specific how this repository is populated and maintained.

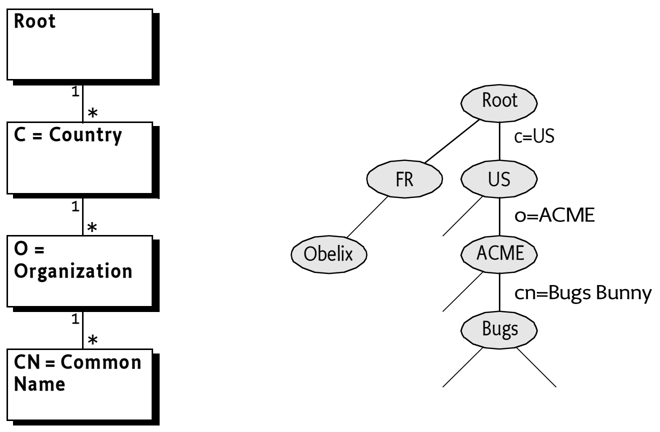
### Sigh... this is a big missing part of the spec. It makes testing hard if not impossible for many cases... I think we should at least have a simple repository.

2.2.4 Distinguished Names

An X.509 name is a *Distinguished Name* (DN) This is a highly structured name, officially identifying a node in an hierarchical name space. The DN concept was developed for the X.500 directory service which envisioned a world wide name space managed by PTTs. Today, the DN is used as an identifier in a local name space, for example a name space designed by an Operator. For example, given a name space that looks like Figure 14, the DN identifying Bugs Bunny looks like:

```
cn=Bugs Bunny,o=ACME,c=US
```

Figure 14 *Country, Company, Person based name space.*



The traversal of the name space is *reversed* from the order in the DN, the first part specifies the least significant but most specific part. I.e. the order of the attribute assertions is significant. Two DNs with the same attributes but different order are different DNs.

In the example, a node is searched in the root that has an attribute c (countryName) with a value that is US. This node is searched for a child that has an attribute o (origanizationName) with a value of ACME. And the ACME node is searched for a child node with an attribute cn (commonName) that has a value "Bugs Bunny".

The tree based model is the official definition of a DN from the X.500 standards. However, in practice today, many DNs contain attributes that have no relation to a tree. For example, many DNs contain comments and copyrights in the ou (origanizationalUnit) attribute.

The DN from an X.509 certificate is expressed in a binary structure defined by ASN.1 (a type language defined by ISO). However, the Distinguished Name is often used in interaction with humans. Sometimes, users of a system have to acknowledge the use of a certificate or an employee of an Operator must grant permissions based on a Distinguished Name of a customer. It is therefore paramount that the Distinguished Name has a good human readable string representation. The expressiveness of the ASN.1 type language makes this non trivial. This specification only uses DN strings as defined in [4] *RFC 2253* with a number of extensions that are specified by the javax.security.auth.x500.X500-Principal class in CANONICAL form.

However, the complexity of the encoding/decoding is caused by the use of rarely used types and features (binary data, multi valued RDNs, foreign alphabets, and attributes that have special matching rules). These features must be supported by a compliant implementation but should be avoided by users. In practice, these features are rarely used today.

### Can we not just be bold and deny these complicating features? I.e. restrict the set of attributes to the list? Don't think we lose anything.

The format of a string DN is as follows:

```
dn        ::= rdn ( ',' rdn ) *
rdn       ::= attribute ( '+' attribute ) *
attribute ::= name '=' value
name      ::= readable | oid
oid       ::= number ( '.' number ) *
readable  ::= <see table further on>
value     ::= <escaped string>
```

### THis is kind of weird because the comparison is defined by the attribute type ...

Spaces before and after the separators are ignored, spaces inside a value are significant but multiple embedded spaces are collapsed into a single space. The following characters must be escaped with a back slash:

comma	','	\u002C
plus	'+'	\u002B
double quote	'"'	\u0022
back slash	'\'	\u005C
less then	'<'	\u003C
greater then	'>'	\u003E
semicolon	';'	\u003B

Backslashes must already be escaped in Java strings, requiring 2 backslashes in Java source code. For example:

```
DN:             cn = Bugs Bunny, o = ACME++, C=US
Canonical form: cn=bugs bunny,o=acme\+\+,c=us
Java Form:      s = "cn=bugs bunny,o=acme\\+\\+,c=us";
```

The full unicode character set is available and can be used in DNs. String objects must be normalized and canonicalized before being compared.

```
DN:             cn = Bugs Bunny, o = Ð Þ, C=US
Canonical form: cn=bugs bunny,o=đ þ,c=us
Java Form:      s = "cn = Bugs Bunny, o = Ð Þ, C=US";
```

### There is something weird with the encoding ... Á and Ç are translated to a?c? ... bt the icelandic characters are find???

The names of attributes (attributes types as they are also called) are actually translated to an Object IDentifier (OID). An OID is a dotted decimal number, like 2.5.4.3 for the cn (commonName) attribute name. It is therefore not possible to use any attribute name, only the ones in the following list are applicable (in short or long form):

### Again, lets limit ourselves to this list

commonName	cn	2.5.4.3 ITU X.520
surName	sn	2.5.4.4
countryName	c	2.5.4.6
localityName	l	2.5.4.7
stateOrProvinceName	st	2.5.4.8
organizationName	o	2.5.4.10
organizationalUnitName	ou	2.5.4.11
title		2.5.4.12
givenName		2.5.4.42
initials		2.5.4.43

```
generationQualifier                 2.5.4.44
dnQualifier                         2.5.4.46

streetAddress           street          RFC 2256
domainComponent         dc              RFC 1274
userid                  uid             RFC 1274/2798?
```

### Are they the same? One comes from RFC 2798 and the other comes from 1274?? what a mess. Can we limit it? We really need a class to model it.!!

```
emailAddress                            RFC 2985
serialNumber                            RFC 2985
```

### The X.500Principal talks about "T", any idea? Can't find a reference to it.

The following DN:

```
2.5.4.3=Bugs Bunny,2.5.4.10=ACME,2.5.4.6=US
```

Is therefore identical to:

```
cn=Bugs Bunny,o=ACME,c=US
```

The attribute types officially define a matching rule, potentially allowing cases sensitive and case insensitive. The attributes in the previous list all match case insensitive. Therefore, an OSGi DN must not depend on case sensitivity.

The X.500 standard supports multi valued RDNs, however, their is not recommended. See [19] *Understanding and Deploying LDAP Directory Services* for the argumentation of this recommendation. Multi valued RDNs separate their constituents with a plus sign ('+' \u002B). Their order is not significant but must always be alphabetical and then numerical for the OIDs. For example:

```
cn=Bugs Bunny+dc=x.com+title=Manager,o=ACME,c=US
```

2.2.5 Certificate Matching

Certificates are matched by their Subject DN. Before comparison two DNs, they must first be canonicalized according to the algorithm specified in javax.security.auth.x500.X500Principal.

DNs can also be compared using wildcards. A wildcard ('*' \u002A) replaces all possible values. Due to the structure of the DN, the comparison is more complicated than string based wildcard matching.

A wildcard can stand for a number of RDNs, or the value of a single RDN. DNs with a wildcard must be canonicalized before they are compared. This means, among other things, that spaces are ignored, except in values.

The format of wildcard DN match is:

```
CertificateMatch::= dn-match ( ';' dn-match )
dn-match        ::= trusted-match | normal-match
trusted-match   ::= '<' normal-match '>'
normal-match    ::= rdn-match ( ',' rdn-match ) * | '-'
rdn-match       ::= '*' | name '=' value-match
value-match     ::= '*' | value-star
```

```
value-star      ::= < value, requires escaped '*' and '-' >
```

### How do we handle escaping of the '*' in value?

The most simple case is a single wildcard, it must match any DN. A wildcard can also replace the first list of RDNs of a DN. The first RDNs are the least significant. This list of matched RDNs must not be empty.

For example, a DN with a wildcard that matches all descendant nodes from the ACME node from Figure 14 on page 22, looks like:

```
*, o=ACME, c=US
```

Such a DN with a wildcard matches the following DNs:

```
cn = Bugs Bunny, o = ACME, c = US
ou = Carots, cn=Daffy Duck, o=ACME, c=US
street = 9C\, Avenue St. Drézéry, o=ACME, c=US
dc=www, dc=acme, dc=com, o=ACME, c=US
```

The following DNs must not match:

```
street = 9C\, Avenue St. Drézéry, o=ACME, c=FR
dc=www, dc=acme, dc=com, c=US
o=ACME, c=US
```

If a wildcard is used for a `value` of an RDN, the value must be exactly `*`. The wildcard must match any value, no substring matching must be done. For example:

```
cn=*,o=ACME,c=*
```

This example DN with wildcard must match the following DNs:

```
cn=Bugs Bunny,o=ACME,c=US
cn = Daffy Duck , o = ACME , c = US
cn=Road Runner, o=ACME, c=NL
```

But not:

```
o=ACME, c=NL
dc=acme.com, cn=Bugs Bunny, o=ACME, c=US
```

Both forms of wildcard usage can be combined in a single matching DN. For example, to match any DN that is from the ACME company worldwide, use:

```
*, c=ACME, c=*
```

Matching of a DN takes place in the context of a certificate. This certificate is part of a *certificate chain*, see *Certificates* on page 20. Each certificate has a Subject DN and an Issuer DN. The Issuer DN is the Subject DN of the certificate that was used to sign the first. The matching of a DN can therefore be extended to match the signer. The semicolon (`';' \u003B`) must be used to separate DNs in a chain.

The following example matches any certificate signed by `Tweety Inc.` in the US.

```
* ; ou=S & V, o=Tweety Inc., c=US
```

### I seem to be missing 2 possibilities. First the possibility to match more than one certificate. Second to mark a certificate in the repository. Proposal:

The minus sign ('-' \u002D) represents any list of certificates, where the asterisk only represents a single certificate. For example, to match a certificate where the Tweety Inc. is in the certificate chain, use the following expression:

```
-;*, o=Tweety Inc., c=US
```

The previous example matched if the Tweety Inc. certificate was trusted, or was signed by a trusted certificate. Certain certificates are trusted because they are known by the Framework. This repository can be indicated with less then ('<') and less then ('>') characters in a matching DN. A DN enclosed in these characters must be directly trusted by the Framework. I.e. it must be available in the trusted repository. As a special case, the'<>' string must match any certificate in the repository.

The following example matches any certificate signed by Tweety Inc, but only if that certificate is in the repository.

```
-;<*, o=Tweety Inc., c=*>
```

2.3 References

[4] *RFC 2253*
http://www.ietf.org/rfc/rfc2453.html

[5] *X.509 Certificates*
http://www.ietf.org/rfc/rfc2459.html

[6] *Java Jar Signing*

[7] *Applied Cryptography*

[8] *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

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

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

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

[12] *Secure Hash Algorithm 1*
http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[13] *RFC 1321 The MD5 Message-Digest Algorithm*
http://www.ietf.org/rfc/rfc1321.txt

[14] *RFC 1421 Privacy Enhancement for Internet Electronic Mail*
http://www.ietf.org/rfc/rfc1421.txt

[15] *DSA*
http://www.itl.nist.gov/fipspubs/fip186.htm

[16] *RSA*
http://www.ietf.org/rfc/rfc2313.txt which is superseded by

http://www.ietf.org/rfc/rfc2437.txt

[17] *Public Key Crypography Standard #7*
http://www.rsasecurity.com/rsalabs/node.asp?id=2129

[18] *Unicode Normalization UAX # 15*
http://www.unicode.org/reports/tr15/

[19] *Understanding and Deploying LDAP Directory Services*
ISBN 1-57870-070-1

3 Module Layer

The standard Java platform provides limited support for packaging, deploying, and validating Java-based applications and components. As a result of this, many Java-based projects, such as jBoss and NetBeans, have resorted to creating custom module-oriented layers with specialized class loaders for packaging, deploying, and validating applications and components. The OSGi Framework provides a generic and standardized solution for Java modularization.

3.1 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. The Framework defines a unit of modularization, called a bundle. Bundles are modules that can share Java *packages* between an *exporter* and an *importer* in a well defined way.

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 required functionality. 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. A bundle JAR file can also embed additional JAR files that are available as resources and classes. This is however not recursive.
- 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 *Resolving* on page 37.
- 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.

3.1.1 Bundle 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-Classpath, 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 [28] *Manifest Format*.

All manifest headers are optional for this release, except for the Bundle-ManifestVersion header. Any manifest header that is not specified has no value by default (except for Bundle-Classpath that has dot ('.', \u002E) as default).

All manifest headers that may be declared in a bundle's manifest file are listed in Table 2, "Manifest Headers," on page 30.

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 and directives on OSGi-defined manifest headers.

3.1.2 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.
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 57.
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-Localization	### get rid of it?	

Table 2 Manifest Headers

Header	Sample	Description
Bundle-ManifestVersion	2	Defines that the bundle follows the rules of this specification. If this manifest header is absent, it must be a bundle of a previous release and its headers must be interpreted according to the OSGi Service Platform Specification R3.
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 67.
Bundle-RequiredExecutionEnvironment	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 36.
Bundle-SymbolicName	com.acme.daffy	
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 *Importing a Lower Version Than Exporting* on page 67.
Export-Package	org.osgi.util.tracker	Comma separated list of package names (with optional version specification) that can be exported. See *Export-Package* on page 41.
Export-Service	###	
Fragment-Host	###	
Import-Package	org.osgi.util.tracker, org.osgi.service.io;version=1.4	Comma separated list of package names (with optional version specification) that must be imported. See *Import-Package Header* on page 40
Import-Service	###	
Private-Package	###Proposed!!!	

Table 2 *Manifest Headers*

Header	Sample	Description
ReExport-Package	###	
Require-Bundle	###	

Table 2 *Manifest Headers*

3.1.3 Value Syntax

Each Manifest header has its own syntax. In all descriptions, the syntax is defined with [29] *W3C EBNF*. The following tokens are predefined for convenience.

```
letter       ::= [A-Za-z_]
digit        ::= [0-9]
jletter      ::= /* see [30] Lexical Structure Java Laguage for
                    JavaLetter */
ws           ::= ' ' | #9
token        ::= letter ( letter | digit ) *
identifier   ::= jletter ( jletter | digit ) *
argument     ::= token | '"' ( [^"\r\n] | \" )* '"'
parameter    ::= directive | attribute
directive    ::= token ':=' argument
attribute    ::= token '=' argument
number       ::= digit+
unique-name  ::= identifier ( '.' identifier )*
package-name ::= unique-name
path         ::= path-char+ ( '/' path-char+ )
path-char    ::= (jletter | digit )*
qname        ::= /* See [30] Lexical Structure Java Laguage for
                    fully qualified class names */
```

3.1.3.1 Version

Versions are used in several places. A version is defined as follows:

```
version   ::= major( '.' minor ( '.'
     micro ( '.' qualifier )? )? )?

major     ::= number
minor     ::= number
micro     ::= number
qualifier ::= ( letter | digit | '-' )+
```

Whitespace is not allowed in a version.

3.1.3.2 Version Ranges

A version range describes a range of versions using a mathematical notation. The syntax of a version range is:

Versions of packages and bundles can be selected by specifying a version range. The syntax of a version range is specified using[31] *Interval Notation*

```
version-range ::= interval | version
interval ::= [ '[' | '(' ] floor ',' ceiling [ ']' | ')' ]
atleast ::= version
floor ::= version
ceiling ::= version
```

Ranges must not contain whitespace.

In the following table, for each specified range in the left-hand column, a version *x* is considered to be a member of the range if the predicate in the right-hand column is true.

Example	Predicate
[1.2.3,4.5.6)	1.2.3 <= x < 4.5.6
[1.2.3,4.5.6]	1.2.3 <= x <= 4.5.6
(1.2.3,4.5.6)	1.2.3 < x < 4.5.6
(1.2.3,4.5.6]	1.2.3 < x <= 4.5.6
1.2.3	1.2.3 = x

Table 3 *Examples of version ranges*

Note that the use of a comma in the version range requires it to be enclosed in double quotes for most manifest headers. Example:

```
Import-Package: com.acme.foo;version="[1.23,2)",
```
com.acme.bar;version="[4.0,5.0)"

### I suggest we make the default of a version range [0,inf)

3.1.4 Filter Syntax

Filters are used in many places in the OSGi. Filter allow for a concise description of a constraint.

The syntax of a filter string is based upon the string representation of LDAP search filters as defined in [24] *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's class 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. If the object is not a recognized type, but the object's class has a constructor that takes a String object as argument, the Framework will construct a new object with the value from the filter and the original and constructed objects can then be compared using equals.

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)".

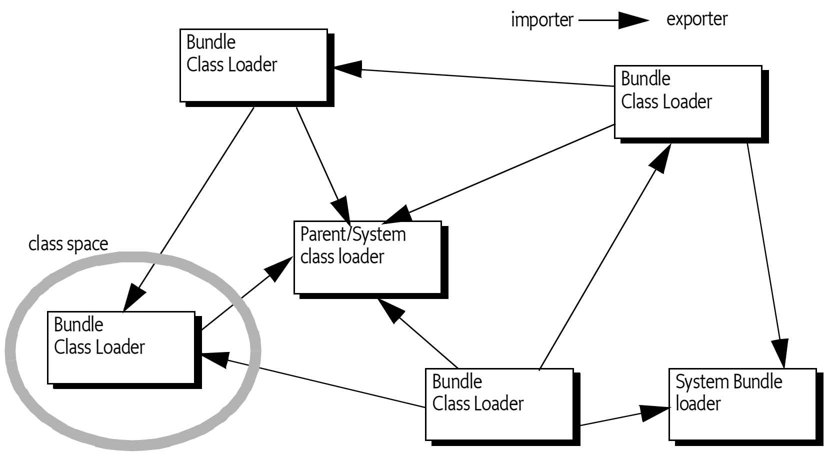
#### Examples

3.2 Class Loading Architecture

A package is uniquely defined by its fully qualified Java name (e.g. java.lang) and a version. By default, all packages are private to a bundle. A bundle must explicitly indicate the packages it is willing to share. This is achieved with the manifest headers Import-Package (*imports*) and Export-Package (*exports*).

A bundle is associated with a unique class loader. That class loader forms a delegation network with other bundles.

Figure 15 Class Loader Delegation model

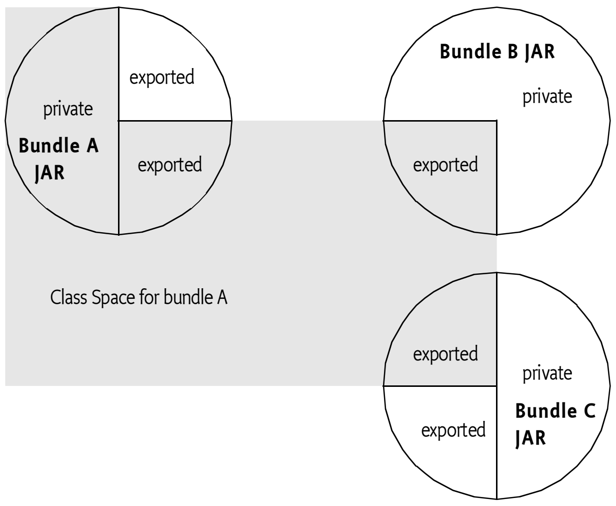


A *class space* is then defined as all classes reachable from a given bundle's class loader. Thus, a class space for a given bundle can contain classes from:

- The parent class loader (normally java.* packages from the system class path)
- Shared packages
- Required bundles
- The bundle's JAR file (*private packages*)
- Fragments

A class space must be consistent in that it *never* contains two classes with the same fully qualified name (to prevent Class Cast Exceptions). However, separate class spaces in an OSGi Platform may contain distinct classes with the same fully qualified name. I.e. the modularization layer supports a model where multiple versions of the same class are loaded in the same VM.

Figure 16 *Class Space*



Class spaces are *compatible* if class names that appear in both class spaces refer to the same class object.

### Yes?

The Framework therefore has a number of responsibilities related to the class loading:

- Before a bundle is started, resolve the constraints that a set of bundles place on the sharing of packages. Then select the best possibilities to create a wiring. This process is extensively described in *Resolving Process* on page 55.
- Start the bundle. This is described in the life cycle layer, ###
- Provide run time class loading according to the calculated wiring using delegation and optional dynamic imports. The runtime aspects are described in *Runtime Class Loading* on page 57.

3.3 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.

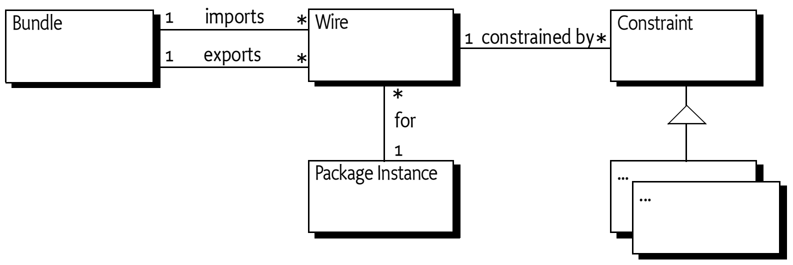
Bundles should list all (known) execution environments that can run the bundle.

3.4 Resolving

The Framework must *resolve* bundles. Resolving is the process where importers and exporters are *wired*. It is a process of constraints solving. This process must takes place before any code from a bundle is executed

A wire is an actual connection between an *exporter* and an *importer*, which are both bundles. A wire is associated with a number of constraints that are defined by its importer's and exporter's manifest headers. A *valid* wire is a wire that has satisfied all its constraints. In Figure 17 it is shown how the class structure of the wiring model looks like.

Figure 17 Class structure of wiring



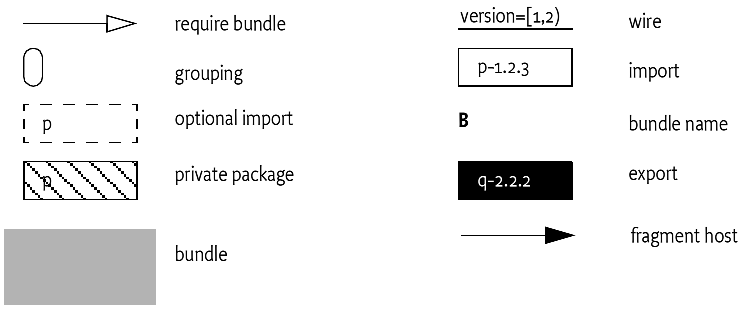
3.4.1 Diagrams and syntax

Wires create a mesh of nodes. Both the wires as well as nodes (bundles) carry a lof of information. In the next sections, the following conventions are used to explain the myriad of details.

Bundles are named A,B, C ... I.e. uppercase characters starting with A. Packages are named p, q, r, s, t ... I.e. lower case characters starting with p. If a version is important, it is indicated with a dash followed by the version: q-1.0. The syntax A.p means the package definition (either import or export) of package p by bundle A.

Import definitions are graphically show by an white box. Export definitions are displayed with a black box. Bundles are a set of connected boxes. Constraints are written on the wires, which are represented by arrows. The point of the arrow of a wire points to the exporter, i.e. it displays the dependency. Figure 19 shows the legend.

Figure 18 *Legend of wiring instance diagrams, and example*

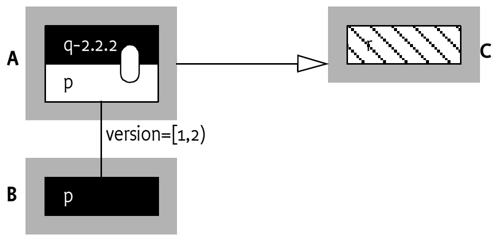


For example:

```
A: Import-Package: p; version="[1,2)"
   Export-Package: q; version=2.2.2
   Require-Bundle: C
B: Export-Package: p; version=1.5.1
```

In Figure 19 it is shown how these definitions could results in a wiring.

Figure 19 *Example bundle diagram*



The following sections define the syntax for the headers that are relevant to the bundle resolving.

3.4.2 Parameters

Many headers use parameters to associate information with objects. Parameters can be either a *directive* or an *attribute*. A directive is instruction to the resolved that cannot be directly matched but has some implied semantics. Attributes can be matched.

### Except for the version attribute ...

3.4.3 Bundle-ManifestVersion

A bundle manifest must express the version of the manifest syntax in which it is written by specifying a Bundle-ManifestVersion header. Bundles exploiting this specification version's syntax (or later) must specify this header. The syntax of this header is as follows:

```
Bundle-ManifestVersion ::= '2'
```

Bundle manifests written to previous specifications manifest syntax are taken to have a bundle manifest version of '1', although there is no way to express this in such manifests. I.e. value of '1' for this header is invalid.

OSGi implementations should support bundle manifests without a Bundle-ManifestVersion header and assume R3 compatibility at the appropriate places. #### reference #### However, an implementation must fail the installation of a bundle with a Bundle-ManifestVersion other then 2.

Version 2 bundle manifests must specify the bundle symbolic name. They need not specify the bundle version as this has a default value.

If the syntax of the bundle manifest does not conform to its manifest version, the bundle must fail to install.

3.4.4 Bundle-SymbolicName

The Bundle-SymbolicName manifest header is a mandatory header. The Bundle Symbolic Name and Bundle Version allow for a bundle to be uniquely identified in the framework. I.e. a bundle with a given symbolic name and version is treated as identical to another bundle with the same symbolic name and version.

The installation of a bundle with a Bundle-SymbolicName and Bundle-Version equal to an existing bundle must fail.

### Yes?

The Bundle-SymbolicName is assigned to a bundle by the developer. The Bundle-Name manifest header provides a human readable name for a bundle and is therefore not replaced by this header.

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

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

The following attribute is recognized by the framework for Bundle-SymbolicName.

### Or a directive?

- singleton – Indicates that the bundle is a bundle that can only have a single version installed. A value of true indicates that the bundle is a *singleton bundle*. The default value is false. When multiple versions of a singleton bundle with the same symbolic name are installed, the framework implementation must select at most one of the singleton bundles to resolve. Only bundles specified as singleton bundles are treated as such. In particular, singleton bundles do not affect the resolution of non-singleton bundles with the same symbolic name.

### Which one? Shouldn't we have a parameter 'shared' Singleton should be the default imho

- fragment-attachment – Related to fragments, see *Fragment-Host* on page 42. If this parameter is always, it 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). If the value is resolve-time, (the default) it indicates that fragments are allowed to attach to the host bundle only during the process of resolving the host bundle. The value never indicates that no fragments are allowed to attach to the host bundle at any time. If a Framework does not support attachment at run time then it should not resolve this bundle.

### This is the only time we have information makes the distinction between runtime and resolve time muddy .... This is soooooo messy. It should only be used at resolve time imho

For example:

```
Bundle-SymbolicName: org.osgi.impl.service.cm;singleton=true
```

3.4.5 Bundle-Version

The Bundle-Version is an optional header, the default is 0.0.0.

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

If the minor or micro version components are not specified, they have a default value of 0. If the qualifier component is not specified, it has a default value of the empty string ("").

Versions are comparable. Their comparison is done nummerically and sequentially on the major, minor, and micro components and last string wise on the qualifier.

A version is considered to be equal to another version if the major, minor and micro components are equal and the qualifier components are equal (using String.compareTo). The Version class can be used to perform this comparison.

Example:

```
Bundle-Version: 22.3.58.build-345678
```

3.4.6 Import-Package Header

The Import-Package header defines the constraints on the imports of the shared package. The syntax of the Import Package header is:

```
Import-Package ::= import ( ',' import )*
import ::= package-names ( ';' parameter )*
package-names ::= package-name ( ';' package-name )*
```

The header allows the definition of many packages. However, an *import definition* is the description for one bundle of a single package.

Import directives are:

- grouping – A list of one or more grouping names separated by commas, default is empty. The argument must be enclosed in double quotes when multiple groups are used. See *Grouping* on page 48.

### Could we use a vertical bar? This would make parsing easier

- resolution – Indicates if the packages must be resolved if the value is mandatory, which is the default. A value of optional indicates that the associated packages are optional. See *Optional Packages* on page 47.

Arbitrary matching attributes may be specified. The following arbitrary matching attributes are predefined:

- version – A version-range to select the (re-)exporter's version. The syntax must follow *Version* on page 32. For more information on version selection, see *Version matching* on page 46.
- bundle-symbolic-name – The bundle symbolic name of the exporting bundle.
- bundle-version – A version-range to select the bundle version of the (re) exporting bundle. The default value is 0.0.0. See *Version matching* on page 46.

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

It is an error that aborts an installation or update when:

- A directive or attribute appears multiple times
- There are multiple import definitions for the same package

Example of a good definition:

```
Import-Package: com.acme.foo;com.acme.bar;
      version=[1.23,1.24];
      resolution:=mandatory
```

3.4.7 Export-Package

The syntax of the Export-Package header is similar to the Import-Package header, only the directives and attributes are different.

```
Export-Package  ::= export ( ',' export )*
export          ::= package-names ( ';' parameter )*
```

The header allows the definition of many packages. However, an *export definition* is the description of a single package.

The following directives

- grouping – The group name for these packages. If no grouping directive is specified, a package will have its own name as grouping name. See *Grouping* on page 48.

### What is the default if the directive applies to multiple packages?

- `mandatory` - A list of attribute names, separated by commas. This argument must be enclosed in double quotes if it contains a comma. A mandatory attribute must be specified by an importer to match. I.e. this overrides the default of always matching an unreferenced (optional) attribute. See *Mandatory Attributes* on page 51.
- `include` – A comma separated list of class, or resource, names that must be visible to an importer. This argument must be enclosed in double quotes if it contains a comma. For class/resource filtering see *Class and Resource Filtering* on page 52.
- `exclude` -A comma separated list of class, or resource, names that must be invisible to an importer. This argument must be enclosed in double quotes if it contains a comma. For class/resource filtering see *Class and Resource Filtering* on page 52.

Arbitrary matching attributes may be specified. The following attribute is part of this specification:

- `version` – The version of the named packages with syntax as defined in *Version* on page 32. It defines the version of the associated packages.

The Framework will automatically associate each package export definition with the following attributes:

- `bundle-symbolic-name` – The bundle symbolic name of the exporting bundle.
- `bundle-version` – The bundle version of the exporting bundle.

It is an error that aborts an installation or update when:

- A directive or attribute appears multiple times
- A package name appears in multiple export definitions.
- The bundle-symbolic-name or bundle-version are specified in the Export-Package header.

An export definition does not automatically imply an import definition. A bundle must explicitly have an import definition for each export definition it requires.

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

Example:

```
Export-Package: com.acme.foo;com.acme.bar;version=1.23
```

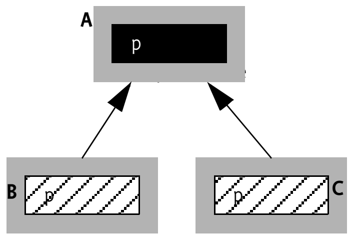
3.4.8 Fragment-Host

A *fragment* is a bundle that is attached to another bundle called the *host bundle*. The components of the fragment, like the Bundle-Classpath, and other definitions are added at the end of the related definitions of a host bundle. All classes and resources within the fragment bundle must be loaded using the class loader of a host bundle.

The Fragment-Host manifest header must conform to the following syntax:

```
Fragment-Host ::= bundle-description
bundle-description ::= unique-name (';' parameter ) *
```

Figure 20 *The fragments B and C are attached to host A*



The following directives are architected by the framework for Fragment-Host:

- multiple-hosts – If the value of this parameter is true then the fragment will attempt attach to all bundles with the given sybolic name and matching the bundle-version range. If the value is false, then the fragment must only attach to the selected bundle with the greatest version that matches the bundle version range, that can be resolved. The default value is false.

### so if we install A-1, F(A,[0,2]) is attached. to A Now we install A-2, F will attach to to A-2 or not. Cant we get rid of this header? it makes things so unpredicatable?

The following attributes are architected by the framework for Fragment-Host:

- bundle-version – The version range to select the bundle that provides the host bundle. See *Version matching* on page 46. The default value is 0.0.0.

### forgetting the dynamic attach ... please? Otherwise we need to extend this we dyanmic attaching

If a bundle specifies the Fragment-Host header, it is illegal to specify any of the following header:

- Bundle-Activator

In order for a host bundle to allow fragments to attach, the host bundle must have BundlePermission[<bundle symbolic name>,HOST]. In order to be allowed to attach to a host bundle, a fragment bundle must have BundlePermission[<bundle symbolic name>,FRAGMENT].

3.5 Linking Bundles (Optional)

The preferred way of linking bundles is to use the Import-Package and Export-Package headers because they couple the importer and exporter to a much lesser extent. However, there are a number of use cases, mainly related to moving legacy code into an OSGi world, that are very cumbersome to implement with the import and export headers.

The Framework can therefore optionally support a mechanism where bundles can be directly linked to other bundles. The following sections define the relevant headers and then discuss the possible scenarios. The Framework must implement all of these mechanisms or none.

A Framework that does not implement these headers must refuse to install or update a bundle that carries these headers. It must throw an exception at install or update time.

### Yes?

3.5.1 Require-Bundle

The Require-Bundle manifest header contains a list of bundle symbolic names that need to be placed on its local classpath. However, only packages that are marked exported are *visible* on this class path.I.e. the exported packages from the required bundles are not imported and no wires are created during resolving but the class loader will search through the list of required bundles to find a class or research.

### There seems to be an inconsistency. The wiring is searched first before dynamic import, required bundles are searched after dynamic import. This makes it very confusing to see require bundle as exports and imports, I think we should just treat it as a classpath .... This would also elegantly solve the split packages. I.e. require bundle is an extension of the Bundle-Classpath, not of import/export definitions.

However, there is an issue with grouping, but this was kind of inconsistent anyway. Maybe grouping should not be supported by require bundle, it is counter intuitive anyway. Complicated!

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

```
Require-Bundle ::= bundle-description
   ( ',' bundle-description )*
bundle-description ::= unique-name (';' parameter )*
```

### It is not very consistent that bsn cannot be repeated like an import/export definition. It makes just as much sense.

The following directives can be used in the Require-Bundle header:

- visibility – A private (default) value indicates that all visible packages from the required bundles are not re-exported. If the value is reexport then all the visible packages of the required bundles are exported from this bundle as if they were local to this bundle.
- resolution – If the value is mandatory (default) then the required bundle must exist for this bundle to resolve. If the value is optional, the bundle will resolve even if the required bundle does not exist.

The following matching attribute is architected by the Framework:

- bundle-version – The value of this parameter is a version-range to select the bundle version of the required bundle. The default value is 0.0.0, implying all versions.

A given package may be visible from more than one of the bundles that are required. This is explicitly allowed, these packages are treated as *split packages*. A split package is package where its contents can come from multiple bundles. Resources and classes from a split package must be searched in the order in which the required bundles are specified in the Require-Bundle header.

### Isnt there a problem that 2 bundles with a different order and overlapping contents of a package get into problems?

### What do we do when A reexports B.p, but not C.p?

### what do we do with A reexport B.p-1 and C.p-2?

As an example, assume that a bundle is split in two bundles and a number of language files that are optional.

```
Require-Bundle:  com.acme.façade;visibility:=reexport,
 com.acme.bar.one;visibility:=reexport,
 com.acme.bar.two;visibility:=reexport,
 com.acme.bar._nl;visibility:=reexport;resolution:=optional,
 com.acme.bar._en;visibiilty:=reexport;resolution:=optional
```

A bundle may both import packages (via Import-Package) and require one or more bundles (via Require-Bundle), but if a package is imported via Import-Package it is not also imported via Require-Bundle: Import-Package takes priority over Require-Bundle and packages which are exported by a required bundle and imported via Import-Package must *not* be treated as split packages.

In order to be allowed to require a name bundle, the requiring bundle must have BundlePermission[<bsn>, REQUIRE_BUNDLE].

3.5.2 Reexport-Package

The ReExport-Package header defines which packages that are obtained from required bundles or from imported packages are re-exported by the bundle. The header can redefine the directives and attributes of the original definition. The purpose of this header is to allow the bundle to act as a façade for a number of other bundles.

The syntax of the header is identical to the syntax of the Export-Package header, see *Export-Package* on page 41, with the following differences:

- The same package may be re-exported more than once with different parameters.
- Filtering, the include and exclude directives, are not supported.

### I do not understand why we allow multiple on ReExport-Package and not on Export-Package? Also, allow filtering as well for symmetry reasons.

### What happens when the parameters are identical?

Re-exported packages can impose multiple group constraints on importers as a grouping can be specified by the re-exporter as well as by the original exporter. See *Grouping* on page 48 for more about grouping.

### I Need a use case for this

### This needs more elaboration here so we can ignore it in other places. I am not sure I fully understand the implications of this.

Any bundle which attempts to re-export a package which is not imported or found in a required bundle must fail to resolve.

The re-export statement must not inherit attributes or directives from the imported packages. If these are not specified on the re-export statement, they are either unspecified or take default values where defaults are defined.

### Unspecified is not acceptable, default is ok.

### This seems to be contradictory with grouping? It was said earlier that they inherit.

3.6 Constraint Solving

The OSGi Framework resolver provides a number of mechanisms to match the importers to the exporters. The following sections describe these sections in detail.

3.6.1 Version matching

Version constraints are a mechanism whereby an import definition can declare a precise version or an arbitrary version range that is acceptable for resolving a package dependency.

Version ranges encode the assumptions about compatibility. The most common compatibility rule is that a change in the:

- major – An incompatible update
- minor – A backward compatible update
- micro – A change that does not affect the interface, i.e. a bug fix

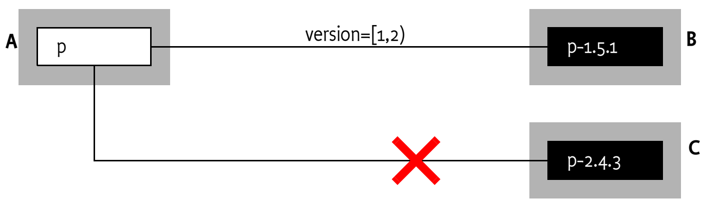
An import definition must specify the version with the version range as the value for the version attribute and the exporter must specify a version as the value for the version attribute. Matching is done with the rules for version range matches as described in *Version Ranges* on page 32.

For example, the following import and export definition resolve correctly because the version range in the import matches the exported version:

```
A: Import-Package: p;
        q;
        version="[1,2)"
B: Export-Package: p;
        q;
        version=1.5.1
```

Figure 21 graphically shows how a constraint can exclude an exporter.

Figure 21 *Version Constrained*



3.6.2 Optional Packages

A bundle can indicate that it does not require a package but it could use the pckage when it was available. For example, logging is important, but the absence of a log service should not prevent a bundle from running. It is therefore possible for an importer to specify that it can match a package even if it does not exist in the system. I.e. so called optional packages.

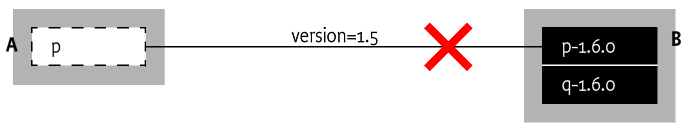
The `resolution` directive of the import definition can take the value `mandatory` (default) or `optional`.

- `mandatory` – indicates that the package must be wired for the bundle to resolve.
- `optional` – indicates that the importing bundle may resolve without the package being wired.

The following example will resolve even though B does not provide the correct version (the package will *not* be available to the code regardless that A resolves).

```
A: Import-Package: p;
        resolution:=optional;
        version=1.5
B: Export-Package: p;
        q;
        version=1.6.0
```

Figure 22 *Optional import*



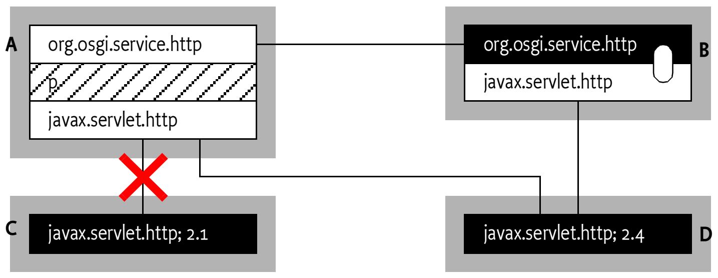
The implementation of a bundle that uses optional packages must be prepared to handle the fact that the packages can be absent. I.e. a VM can throw an exception when there is a reference to a class in a missing package.

3.6.3 Grouping

Package grouping specifies a dependencies between packages. It is used to ensure class space consistency by indicating that the import of a package *implies* the use of other packages. For example, javax.servlet.http implies the package javax.servlet because javax.servlet.http.HttpServlet extends a class from the package javax.servlet. Class space consistency can only be ensured if a bundle has only one exporter for each package, either implied or imported.

For example, the Http Service implementation requres servlets to extend the javax.servlet.http.HttpServlet base class. If the Http Service bundle would import version 2.4 and the client bundle would import version 2.1 then a class cast is bound to happen. This is depicted in Figure 23.

Figure 23 Grouping in B, forces A to use javax.servlet from D

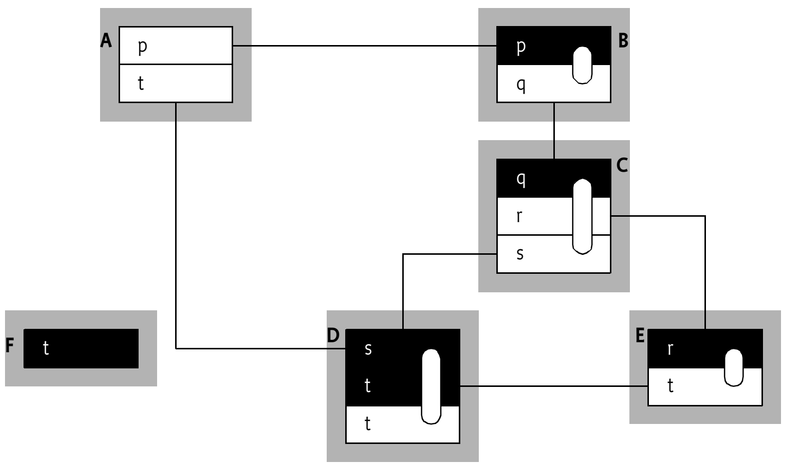


If a bundle imports a package from an exporter then the export definition of that package can imply a number of other packages through *grouping*. These *implied packages* must be wired to an exporter that in its turn can imply packages, ad nauseum. I.e. a single import of a package can imply a large set of other packages from many different bundles. The term implied packages refers to the complete set of implied packages from recursively traversing the wires. Implied packages are not imported, an implied package can only constrain an import definition.

For example, in Figure 24, bundle A imports package p. Assume this import definition is wired to bundle B. Due to the grouping (the ellipse symbols groups the packages) this implies package q.

Further assuming that the import for q is wired to C, then this implies package r and s. Continuing, assuming C.s and C.r are wired to bundle D and E respectively. These bundles both add t to the set of implied packages for bundle A. Last, assuming that D and E are wired to F mandates that A is also wired to F for package t.

Figure 24 *Implied Packages*



The implied packages for the bundles are therefore:

	A	B	C	D	E
p					
q	x				
r	x	x			
s	x	x			
t	x	x	x		

To maintain class space consistency, the Framework must ensure that none of its bundle imports conflicts with any of that bundle's implied packages.

For the example, this means that the Framework must ensure that the import definition of A.t is wired to D.t. Wiring this import definition to F.t violates the class space consistency because A could be confronted with objects with the same class name but from the different classloaders of bundle G and F. which would create class cast exceptions.

The exporter can express the implied packages with the grouping directive on the export definition.

### maybe I am just fooling myself but it feels like the concept of an implied package is easier to understand than the public export, grouping, reexport, etc concepts. It somehow seems more logic to have a implies directive on the export than a grouping directive on both. I.e. it would be easy for a tool to analyze a package and detect its implied packages, while this is impossible to find the reverse from the implied packages.

The grouping directive is private to a bundle, its only purpose is to indicate which packages are implied by an export definition. The grouping directive creates a group that links these together. The group name only expresses the relation between import and export definitions within a *single* bundle. Group names have no meaning between bundles.

An exported package, by default, belongs in a group with its own name. For example, the following definition expresses the dependency that the Http Service requires its clients to use the same servlet package.

```
Export-Package: org.osgi.service.http
Import-Package: org.osgi.service.http;
      grouping=org.osgi.service.http,
      javax.servlet.http;
      javax.servlet;
      grouping:="org.osgi.service.http"
```

A scenario with this case is depicted in Figure 23. Bundle A imports the Http Service classes from bundle B. Bundle B has grouped the org.osgi.service.http and the javax.servlet and A must therefore be wired for javax.servlet to the same root as bundle B.

If multiple packages are involved, then the import definitions can list all the relevant groups in the grouping directive. For example, assume p is used by q and r. The following statements should then assure that any importer that uses q and r will get wired to the same exporter for the package q.

```
Export-Package: p;r
Import-Package: q;
   grouping="p,r"
```

With many exported packages this becomes cumbersome due to the long package names. An optimization is therefore to add the exported packages to a combined group. The next example is identical to the previous example. Wiring to p or q will imply r.

```
Export-Package: p;q;grouping=1
Import-Package: r;grouping=1
```

### This seems to be bad practice and just syntactic sugar (if I understood this thing correct!). Could we not kill the grouping on the export and always use the package names in the import grouping? This is also much easier to automate.

Grouping ensures that the class space of an importer must be consistent with the class space of the exporter for the packages concerned.

Propagated packages can subject importers to multiple grouping constraints since the original exporter (and any re-exporters) may specify additional groupings.

### I need a good example of this case and this seems contradictory with the fact that re-exporting definitions do not inherit ....

As an example of a situation where grouping makes resolving impossible is the following setup.

```
A: Import-Package: q; version=1.0
   Export-Package: p; grouping=q, r
B: Export-Package: q; version=1.0
C: Import-Package: q; version=2.0
```

These constraints cen be resolved because the import A.q can be wired to the export B.q.

An example of a bundle D that cannot be resolved in this constellation:

```
D: Import-Package: p, q; version=2.0
```

D.p must be wired to A.p because A is the only exporter. However, this implies package A.q due the grouping in A. Package A.q is wired to B.q-1.0. Import package D.q requires version 2.0 and can therefore not be resolved without violating the grouping constraint.

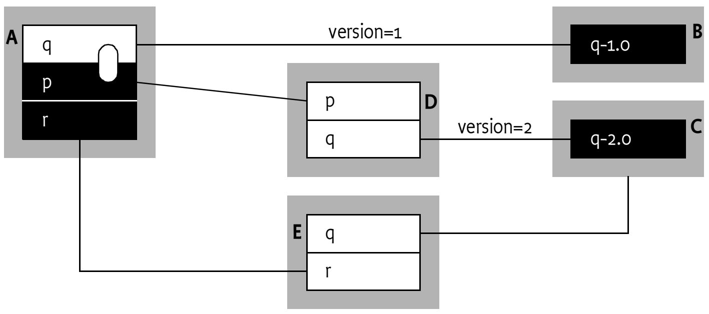
However, consider bundle E:

```
E: Import-Package: r, q; version=2.0
```

Package E.r is wired to A. However, A.r is not grouped and it therefore does not imply any packages. The Framework is therefore free to wire E.q to bundle C.

Both scenarios are depicted in Figure 25.

Figure 25 Grouping and resolving



3.6.4 Attribute Matched Packages

Attribute matching is a generic mechanism to allow the importer and exporter to influence the matching process in a declarative way. Arbitrary attributes specified in the Import-Package of the importer must match the attributes of the exporter's Export-Package statement .

For example, the following statements will match.

```
A: Import-Package: com.acme.foo;company=ACME
B: Export-Package: com.acme.foo;
          company="ACME";
          security=false
```

Attributes are compared string wise except for the version and bundle-version attributes which use range comparisons. Leading and trailing whitespace must be ignored.

3.6.5 Mandatory Attributes

There are two types of attributes: *mandatory* and *optional*. Mandatory attributes must be included in the import definition to match. Optional attributes are ignored when they are not referenced by the importer and therefore do not influence the match. Attributes are by default optional.

The exporter can specify mandatory attributes with the mandatory directive in the export definition. This directive contains a comma separated list of attribute names that must be referenced by the leaf to match.

For example, the next statements must not match because security is a mandatory attribute:

```
A: Import-Package: com.acme.foo;company=ACME

B: Export-Package: com.acme.foo;
          company="ACME";
          security=false;
          mandatory=security
```

3.6.6 Class and Resource Filtering

A root can limit the visibility of the resources and classes in a pacakge with the include and exclude directives on the export definition.Each of these directives can hold a comma separated list of class names and/or resource names. These names are without the package prefix, path separators, and in the case of class names with the .class extension. Names can end in an asterisk ('*', \u002A) to indicate a wild card. I.e. any name that matches the part before the asterisk will match.

The default for the include directive is '*' (wildcard matching all names) and for the exclude directive that no classes or resources are excluded, i.e. an empty list that matches no names. If include or exclude directive are specified, the corresponding default is overridden.

### Spec is simpler if we force the user to specify the .class extension. I.e. we now have a kind of odd situation with resources ending in .class, feels asymmetric

### Also, it may also be easier to allow full path names so that multiple packages can be specified in one export definition

Only the resources and classes are visible that are:

- Matched with an entry in the included list, *and*
- Not matched with an entry in the excluded list

In all other cases, loading or finding fails and a Class Not Found exception is thrown for a class load or a null value is returned for a resource load. The ordering of includes and excludes is not significant.

The following example shows and export statement and a list of files with their visibility.

```
Export-Package: com.acme.foo; include="Qux*";
        exclude=QuxImpl

com/acme/foo
   QuxFoo.class          visible    i & !e
   QuxBar.class          visible    i & !e
   Qux.html              visible    i & !e
   QuxImpl.class                    i &  e
   BarImpl.class                   !i & !e
   properties.prop                 !i & !e
```

### Is the filtering also done for the bundle that exports the package? This feels kind of weird. I.e. assume A and B export/import this package. Depending on resolution, A then sees the content or not. Looks error prone?

Care must be taken when using filters. For example, a new version of a module which is intended to be backward compatible with an earlier version should not filter out classes or resources which were not filtered out by the earlier version. Also, when modularizing existing code, filtering out classes or resources from an exported package may break users of the package.

```
include ::= content
exclude ::= content
content ::= content-name ( ',' content-name )*
content-name ::= identifier
   | '*'
   | identifier '*'
```

Class names must not include their package and do not end with .class. I.e. the class com.acme.foo.Daffy is named Daffy in either list. Resource filenames do not include a path name and can end with any file extension, e.g. index.html. Both class names and resource names may end with a single '*' wild card. A name without a period can match both class names and resource names.

3.6.7 Provider Selection

Provider selection allows the importer to select which bundles can be considered as exporters. Provider selection is used when there is no specification contract between the importer and the exporter. The importer tightly couples itself to a specific exporter, typically the bundle that was used for testing. The importer can optionally specify a range of bundle versions that will match to make the wiring less brittle.

An exporter can select an importer with the import attributes bundle-symbolic-name and bundle-version. The Framework automatically provides these attributes for each export definition. These attributes must not be included in the export definition.

The export definition bundle-symbolic-name attribute will contain the bundle symbolic name as specified in the Bundle-SymbolicName header without the parameters. The export definition bundle-version attribute is set to the value of the Bundle-Version header or its default of 0.0.0 when absent.

The bundle-symbolic-name is matched as an attribute. The bundle-version attribute is matched using the version rules as defined in *Version Ranges* on page 32. The import definition must be a version range and the export definition is a version.

For example, the following definitions will match:

```
A: Import-Package: com.acme.foo;
           bundle-symbolic-name=B;
           bundle-version="[1.41,2.0.0)"

B: Bundle-SymbolicName: B
```

```
    Bundle-Version: 1.41
    Export-Package: com.acme.foo
```

The following statements will not match because B does not specify a version and thus defaults to 0.0.0:

```
A: Import-Package: com.acme.foo;
           bundle-symbolic-name=B;
           bundle-version="[1.41,2.0.0)"

B: Bundle-SymbolicName: B
   Export-Package: com.acme.foo;version=1.42
```

Selecting an exporter by symbolic name can result in brittleness because of hard coupling of the package to the bundle. For example, if the exporter eventually needs to be refactored into multiple separate bundles, all importers must be changed. Other arbitrary matching attributes do not have this disadvantage as they can be specified independently of the exporting bundle.

The problem of the brittleness of bundle symbolic name in bundle refactoring can be overcome by writing a façade bundle with the same bundle symbolic name as the original bundle, but this requires a framework which has opted to implement `Reexport-Package`.

3.6.8 Fragments

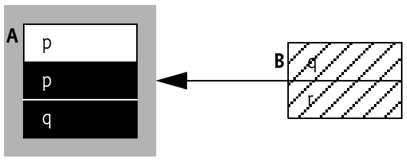
Fragments are bundles that are *attached* to a *host bundle* by the Framework. Attaching is done before resolving, the Framework appends the relevant definitions to the host's definitions before the host is resolved. Fragments are therefore treated as part of the host, they do not have their own classloader.

A key use case for fragments is providing translation files for different locales. This allows the translation files to treated and shipped independently from the main application bundle.

For example, the following definition show how a fragment can be setup (depicted in Figure 26):

```
A: Import-Package: p
   Export-Package: p, q
B: Fragment-Host: A
   Private-Package: q, t
```

Figure 26 Fragment-Host relation



Isnt it better never to resolve it? Seems a bit odd that a fragment is resolved but there is no classloader associated with it.

Updating an attached fragment bundle must cause the host bundle to re-resolved and all attached fragment bundles to reattach and will result in the host bundle being restarted as if the host bundle itself was updated.

This is after refresh I assume? Seems kind of bad that an update kills another bundle?

When attaching a fragment bundle to a host bundle the Framework must perform the following steps:

1. Append the Import-Package entries for the fragment bundle that do not *conflict* with an Import-Package entry of the host to the Import-Package entries of the host bundle.
   A fragment Import-Package entry conflicts with a host Import-Package entry if it has the same package name and any of its directives or matching attributes are different. If a conflict is found, the fragment is not attached to the host bundle. A Fragment can provide an import statement for a private package of the host. The private package in the host is hidden in that case.

Error prone!

2. Append the Require-Bundle entries of the fragment bundle that do not conflict with a Require-Bundle entry of the host to the Require-Bundle entries of the host bundle. A fragment Require-Bundle entry conflicts with a host Require-Bundle entry only if it has the same bundle symbolic name but a different version range. If a conflict is found, the fragment is not attached to the host bundle.

### Can a fragment point to itself? I guess a Require-Bundle must not link to a fragment..

3. Append the Export-Package entries of a fragment bundle that do not conflict with an Export-Package entry of the host to the Export-Package entries of the host bundle. A fragment Export-Package entry conflicts with a host Export-Package entry if it has the same package name. If a conflict is found, the export from the fragment must be ignored.

Why is this not an error?

A fragment bundle must enter the resolved state only if it has been successfully attached to one or more host bundles.

Resolving seems funny for an activator, also it seems to indicate you could actually use it, which is clearly not true

During runtime, the fragment's JAR is searched after the host's bundle classpath as described in *Fragments During Runtime* on page 61.

3.7 Resolving Process

Resolving is the process that creates a wiring between bundles. Constraints on the wires are statically defined by:

- Import and Export Packages
- Required bundles

Before a bundle is resolved, all its fragments must be attached. The resolving process is then a constraint solving algorithm that can be described in terms of requirements on wiring relations. The following constraints must be satisfied for a wiring to be valid.

The resolving process is a iterative process that searches through the solution space.

If a module's import or export definitions have *both* import and export definitions for the *same* package, then the framework needs to decide which to choose.

It must first try to resolve the overlapping import definition. There are 3 possible outcomes:

- *External* – If this resolves to an export statement in another bundle, then the overlapping export definition in this bundle is discarded.
- *Internal* – If it is resolved to an export statement in this module, then the overlapping import definition in this module is discarded.
- *Unresolved* – There is no matching export definition, this is however a developer error because it means the overlapping export definition of the bundle is not compatible with the overlapping import definition.

A bundle is valid if:

- All its mandatory imports are wired
- All its wires are valid
- All its mandatory required bundles are available and their exports validly wired
- All its re-exports are available from the required bundles

A wire is valid when:

- The importer's version range matches the exporter's version. See *Version matching* on page 46.
- All the importer's attributes must match the attributes of the corresponding exporter. See *Attribute Matched Packages* on page 51
- The importer must specify all mandatory attributes from the exporter. See *Mandatory Attributes* on page 51
- Implied packages refering to the same package as the wire are wired to the same exporter. See *Grouping* on page 48.
- The wire is connected to a valid exporter.

The following list defines the preferences if multiple choices are possible in order of priority:

- A resolved exporter must be preferred over an unresolved exporter.
- An exporter with a higher version is preferred over an exporter with a lower version.
- An exporter with a lower bundle id is preffered over a bundle with a higher id.

### yes? Was not deterministic

A valid bundle can be resolved.

3.8 Activation

### I think we must describe here how the system gets started. The module layer is independent of the life cycle layer. Do we define that a bundle still gets started? Would be nice.

3.9 Runtime Class Loading

Each bundle installed in the Framework must not have an associated class-loader until after it is resolved. After a bundle is resolved, the Frameworks must create one class loader for each bundle that is not a fragment.

One class loader per bundle allows all resources within a bundle to have package level access to all other resources in the bundle within the same package. This class loader provides each bundle with its own name-space, to avoid name conflicts, and allows resource sharing with other bundles.

This class loader must use the wiring as calculated in the resolving process to find the appropriate exporters. If a class is not found in the imports, additional headers in the manifest can control the searching of classes and resources in additional places.

The following sections define the headers that influence the runtime class loading and then define the exact search order the Framework must following when a class or resource is loaded.

3.9.1 Bundle Classpath

Intrabundle classpath dependencies are declared in the Bundle-Classpath manifest header. This declaration allows a bundle to declare its *embedded* classpath using one or more JAR files that are contained in the bundle's JAR file.

The Bundle-Classpath manifest header is a list of comma-separated file names. A file name can be either:

- The dot ('.' \u###) represents the bundle's JAR file itself is the default value if no Bundle-Classpath is specified.
- A path to a JAR file contained in the bundle's JAR file.
- A path to a directory contained in the bundle's JAR file.

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

```
Bundle-Classpath::=  entry ( ',' entry )*
entry           ::= target ( ';' target )*
      ( ';' parameter ) *
target          ::= path  |  '.'
```

The following Bundle-Classpath attribute is recognized by the Framework:

Why not a directive?

- selection-filter – A filter that is used to match against system properties values. The corresponding element is only included in the classpath if the filter evaluates to true. The value of this parameter must normally be a quoted-string due to conflicts between the filter syntax and the header syntax.
  If the filter's syntax is invalid, the Framework must publish a Framework Event of type ERROR with the Invalid Syntax Exception and the filter is considered to evaluate to false.

The framework must ignore any unrecognized parameters.

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 filter selection can be used to select only libraries that are necessary on a specific platform. For example, assume the property osname is used to select optimized libraries for different operating systems.

```
Bundle-Classpath: unix.jar;selection-filter="(osname=unix)",
   other.jar; selection-filter="(!(osname=unix))"
```

The Framework must ignore an target in the Bundle-Classpath header if the target (directory or JAR file) cannot be located. However, in that case the Framework should publish a Framework Event of type INFO with an appropriate message for each entry that cannot be located. The Framework should also publish a Framework Event of type INFO if the Bundle-Classpath evaluates to empty.

### Is empty before or after fragments are included.?

### hmmm, missing entries are common with the fragement case ... dont we create a lot of INFO records?

When locating a classpath entry in a bundle the Framework must attempt to locate the classpath entry relative to the root of the bundle's JAR. If a classpath entry cannot be located in the bundle, then the Framework must attempt to locate the classpath entry in each of the attached fragment bundles. The attached fragment bundles are searched in ascending bundle id order. This allows a fragment to supply entries that are inserted into the host's specified Bundle-ClassPath.

The following example illustrates this:

```
A: Bundle-SymbolicName: A
   Bundle-ClassPath: required.jar,optional.jar,default.jar
   content
   required.jar
   default.jar
B: Bundle-SymbolicName: B
   Bundle-ClassPath: fragment.jar
   Fragment-Host: A
   content
   optional.jar
```

```
fragment.jar
```

In this example, bundle A has a Bundle-ClassPath with three entries (required.jar, optional.jar, and default.jar). The required.jar classpath entry may contain the classes and resources that must be present for the bundle to function. The optional.jar classpath entry may contain classes and resources that the bundle will use if present.

The default.jar classpath entry may contain classes and resources that the bundle will use if the optional.jar is not available, but the classes and resources from default.jar may get overridden by classes and resources in the optional.jar (from the fragment) classpath entries. Bundle A has only the required.jar and default.jar entries packaged with it. This allows a fragment bundle B to be installed that can supply the optional.jar for bundle A.

The fragment bundle B has a Bundle-ClassPath with one entry (fragment.jar). When bundle A is resolved and the fragment bundle B is attached then the bundle classpath for the bundle A must be

```
required.jar, optional.jar, default.jar, fragment.jar.
```

3.9.2 Dynamic-ImportPackage

Dynamic imports are matched to exports (to form package wirings) during *class loading* and therefore do not affect module resolution. Dynamic imports apply only to packages which have no import definition. Dynamic import, like Import-Package, takes precedence over Require-Bundle.

### Is this last sentence true? Require Bundle is resolve time isnt it?

```
DynamicImport-Package ::= dynamic-description
          ( ',' dynamic-description )*
dynamic-description::= wildcard-names ( ';' parameter )*
wildcard-names     ::= wildcard-name ( ';' wildcard-name )*
wildcard-name      ::= package-name
        | ( package-name '.*' )
        | '*'
```

No directives are architected by the framework for DynamicImport-Package. Arbitrary matching attributes may be specified. The following arbitrary matching attributes are architected by the framework:

- version – A version range to select the exporter's implementation version. The default value is 0.0.0 .
- bundle-symbolic-name – The bundle symbolic name of the exporting bundle.
- bundle-version – a version range to select the bundle version of the exporting bundle. The default value is 0.0.0.

Packages may be named explicitly or by using wild-carded expressions such as org.foo.* and *. Bundles can not dynamically import the default package.

Dynamic imports must be searched in the order in which they are specified. The order is particularly important when wilcarded package names are used. The order will then determine the order in which matching occurs. If multiple packages need to be dynamically imported with identical parameters, the syntax permits a list of packages, separated by semicolons, to be specified before the parameters.

During class loading, the package of the class being loaded is compared against the specified list of (possibly wild-carded) package names. Each matching package name is used in turn to attempt to wire to an export using the same rules as Import-Package (except for the propagation). If a wiring attempt is successful, the search is forwarded to the exporter's class loader where class loading continues. The wiring must not subsequently modified, even if the class cannot be loaded. This implies that once a package is dynamically resolved, subsequent attempts to load classes or resources from that package are treated as normal imports.

In order for a DynamicImport-Package to be resolved to an export statement, all mandatory arbitrary attributes (as specified by the exporter, see *Mandatory Attributes* on page 51) must be specified in the dynamic import definition. All attributes of the dynamic import definition must match the attributes of the export statement.

### Oh so much rope to hang oneself. The fact that we MAY specify this crap, does not mean that we MUST?

### I need a use case where attributes and versions on Dynamic Import make sense

Dynamic imports are very similar to optionally importing packages, see *Optional Packages* on page 47. They differ in two ways:

- *Resolve time* – Optional packages are a resolve time concept. Dynamic imports are handled during actual class loading.
- *One shot* – Optional imports are never recalculated after resolve. For example, an optional package can not be wired during resolve because there is no exporter. This import definition must remain unresolved even if an exporter becomes available. In contrast, dynamic imports must be tested every time a class is loaded and no wiring exists. I.e. if a wire cannot be constructed the first time, it could still be constructed another time.

### We do not address one of the key issues. Dynamic Import is usually used with class for name. Typically, the name comes from some kind of configuration parameter like a system property or a file. This means that the programmer is delegating the class selection to the end user and NOT the programmer. This use case is at odds with the programming model. here. Only the programmer can statically set the attributes in the manifest (which is useless imho)! I think we need a syntax where the end user can set these attributes in the string:

com.acme.plugin.MyPlugin;version=3.1;bundle-symbolic-name=com.acme.mybundle

Alternatively, I am very happy with a good use case that shows why we need the attributes on the Dynamic Import in the manifest ….

3.9.3 Fragments During Runtime

All class or resource loading of a fragment is handled through the host's class loader, a fragment must never have its own class loader. I.e. he fragment is treated as if it is an intrinsic part of its host.

Fragments must be searched after a host could not locate the class or resource on it's bundle classpath. When the host searches the fragment's jAR , it must obey the fragment's bundle classpath. The fragments are *searched* and this implies that packages can be split over the host and any of its fragments.

Searching the fragments must be done in the the order that the fragment bundles were installed. That is ascending bundle id order.

More deterministic would be if they were searched in name order or so. This is so hard to test

Figure 27 *Resource/Class searching with fragments*

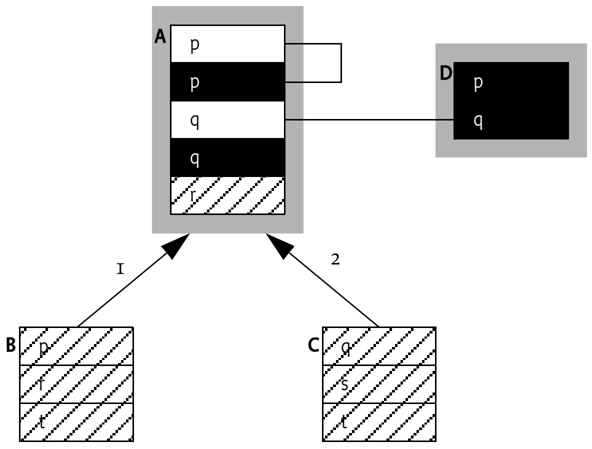


Figure 27 shows a setup with 2 fragments. Bundle B is installed before bundle C and both B and C attach to A. The following table shows where different packages orginate from in this setup. Note that the order of the append (>) is significant.

Package Requested	From	Remark
p	A.p > B.p	if A.p was wired to D.p, then B.p was not searched
q	D.q	if A.q had been wired to A.q, then C.q had been visible.
r	A.r > B.r	

Table 4 *Effect of fragments on searching*

Package Requested	From	Remark
s	C.s	
t	B.t > C.t	

Table 4 *Effect of fragments on searching*

Fragments must remain attached as long as the host remains resolved. When a host bundle becomes unresolved then all attached fragment bundles must be detached from the host bundle. When a fragment bundle becomes unresolved the Framework must detach it from the host and reresolve the host bundle and reattach the remaining attached fragments.

### How can a fragment become unresolved? Its resolve state depends if it is attached or not?

A framework implementation can optionally support attaching fragments to an already resolved bundle if the following conditions are true:

How can I figure out if this framework supports this or not?

### YACCCCCCK! There goes our testing. Can we kill this?

- The fragment is logically at the end of the list of attached fragment bundles of the host bundle. This would normally be the case given that fragment bundles are attached in ascending bundle id order.
- The fragment's Import-Package and Export-Package entries do not add additional packages to the host.
- The fragment's Require-Bundle entries do not add additional required bundles to the host.
- The host allows dynamic attachment because the fragment-attachment directive specifies always.

3.9.4 Overall Search Order

Frameworks must adhere to the following rules for class or resource loading, in the given order. The search must stop when the resource is found.

1 If the class or resource is in a java.* package, the request must be delegated to the parent class loader.

### I think we have not well defined what delegate means. Delegate is not the normal search path. I.e. it looks like it enters step 4?

2 If the requested class or resource is in an imported package, then the request must be delegated to the exporter's class loader.
3 If the requested class matches a package from the Dynamic-Import-Package header, then the Framework must attempt to create a wire to an exporter taking the resolving rules into account. If this succeeds, the request must be delegated to the exporter as in step 2. Otherwise the request must fail.
4 All exported pacakages of the required bundles must searched by consulting the required bundle's classloaders in the order in which they are specified in this bundle's Require-Bundle header. Bundles that are

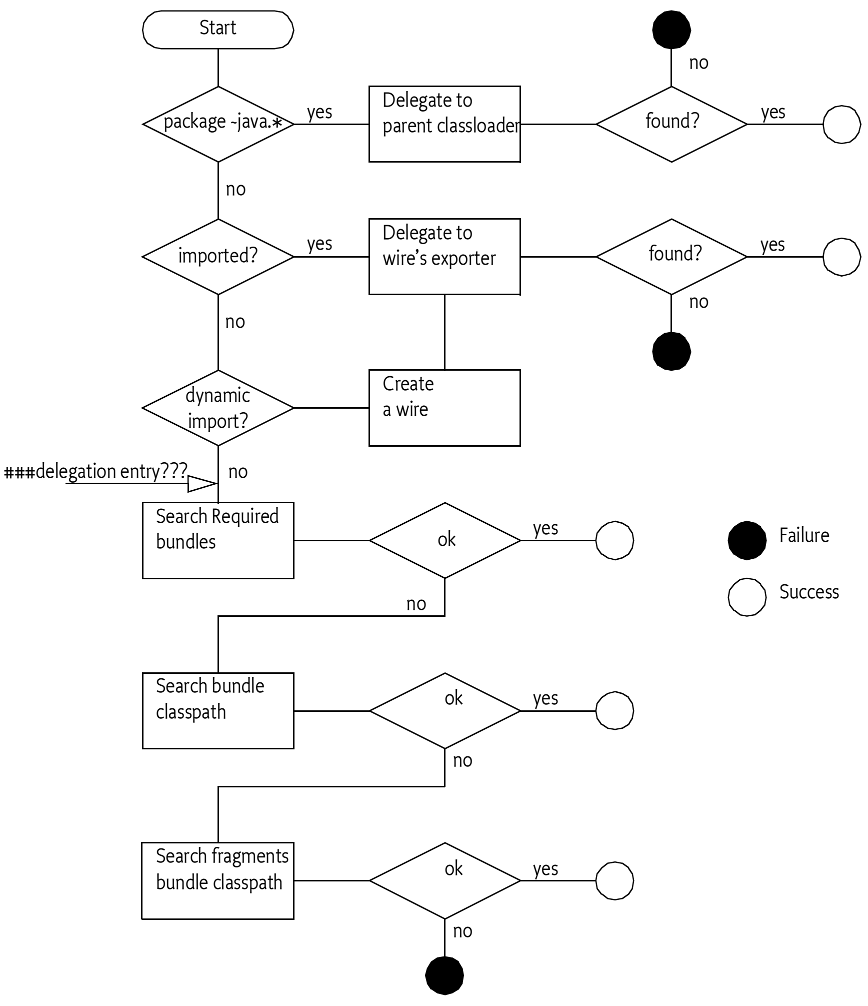
already visited in this request must not be visited again to prevent cycles. See *Bundle Cycles* on page 65 for more information..

### Export Packages from required bundle can not be treated as import packages??

5 The bundle's own bundle class path is searched in the order of the Bundle-Classpath header.
6 Search fragments using the fragment's Bundle-Classpath header into account.

The following non-normative flow chart illustrates the search order described above:

Figure 28 *Flow chart for class loading (non normative)*



3.9.5 Parent class loader

The set of implicitly imported class path packages is defined as all java.* packages, since these packages are required by the Java runtime and using multiple versions at the same time is not easily possible. For example, all objects must extend the same Object class.

A bundle must not declare imports or exports for java.* packages; doing so is an error and any such bundle must fail to install. All other packages on available through the parent class loader must be hidden from executing bundles.

However, the framework must be able to explicitly export packages from the parent class loader. The system property org.osgi.framework.system.packages can contain an export definition for the system bundle. This property employs the standard export manifest header syntax:

### I have changed it to org.osgi.framework.system.packages because this was the property used by the old ref impl.

```
org.osgi.framework.system.packages ::= package-description (
',' package-description )*
```

The system bundle (bundle id is 0) is used to export non-java.* packages from the parent class loader. Export definitions from the system bundle are treated like normal exports, meaning that they may have version numbers, can be grouped, and are used to resolve import package declarations as part of the normal bundle resolving process. Other bundles may provide alternative implementations of the same packages.

The value of this property is set either manually or is calculated by a framework. The exported packages must have a bundle symbolic name value of "system.bundle" and a bundle version value of 0. Exposing packages from the parent class loader in this fashion must also take into account any grouping requirements of the underlying packages. For example, the definition of javax.servlet must be grouped with javax.servlet.http.

3.9.6 Resource Loading

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 must be 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 must be 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.

3.9.7 R3 Compatibility

### I think we should not support this. When do we ever kill it? Now is the time because we had a bad previous spec. By creating a compatibility mode that is well defined we upgrade it to correct behavior.

Since previous versions of the service platform specification were incomplete and/or contradictory regarding the visibility of packages provided via the parent class loader, it is possible that legacy bundles exist that violate the rules defined above. To smooth this transition with legacy bundles, a framework may modify its behavior slightly for bundles that do not contain the Bundle-ManifestVersion header.

A framework is allowed to install legacy bundles that import or export java.* packages. For such legacy bundles, the class and resource search algorithm is modified in two places.

### It is no problem for the new framework to accept java.* ... why not just allow it? Can't harm can't it?

1 The parent class loader must be consulted after step 4. This is after consulting the required bundles and before the bundle's class path is searched. This allows the bundle to load classes and resources directly from the class path, which will allow bundles that expected this behavior to continue to function properly. This algorithm must only be used for R3 or lower legacy bundles; R4 bundles must conform to the strict definition.

### BJ wants this to be like R3, first consult parent for all.

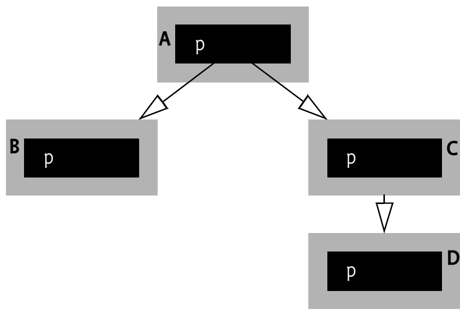
3.9.8 Bundle Cycles

Multiple required bundles can export the same package. Bundles which export the same package that are involved in a require bundle cycle can lead to lookup cycles when searching for classes and resources from the package. Consider the following definitions:

```
A: Require-Bundle: B, C
C: Require-Bundle: D
```

These definitions are depicted in Figure 29.

Figure 29 *Depth First search with Require Bundle*

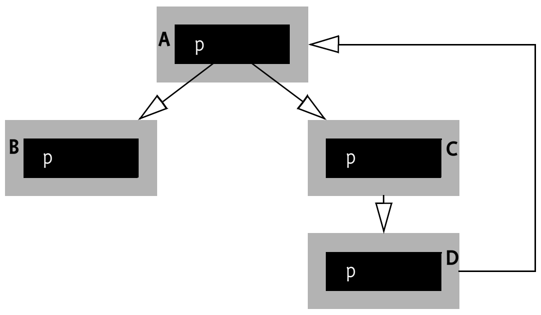


Each of the bundles exports the package p. In this example, A requires B,and C and C requires D. When A is requested to load a class or resource from package p, then the required bundle search order is the following: B, D, C, A. This is a depth first search (DFS) order. The depth first search order can introduce endless search cycles if the dependency tree has a cycle in it.

Using the previous setup, a cycle can be introduced if class space C requires A like the following (and shown in Figure 30):

```
D: Require-Bundle: A
```

Figure 30 *Cycles*



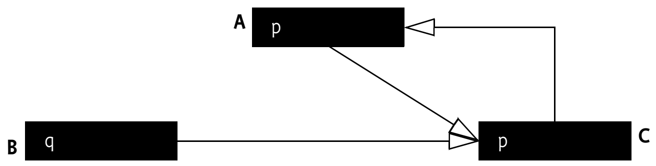
When the classloader for bundle A is requested to load a class or resource from package p then the bundle search order must be the following: B,D, A, B, D, A, ...

Since a cycle was introduced each time bundle D is reached the search will recurse back to A and start over. The framework must prevent such dependency cycles from causing endless recursive lookups.

To avoid endless looping, the framework must mark each bundle upon first visiting it and not explore the required bundles of a previously visited bundle. Using the visited pattern on the dependency graph above will result in the following bundle search order: B, C, D, A.

### This looks very error prone ... It means that the view from C is different when it is accessed from B or A. I.e. in the following example A<p== C.p and B<p == A.p. I think the cycles need to be resolved during resolve time and not runtime. Maybe even rejected.

Figure 31 *Cycles*



3.9.9 Importing a Lower Version Than Exporting

#### rework necessary, but like to keep

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.

3.9.10 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.

3.9.11 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.

### rework

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.

3.10 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
      ( ',' nativecode )* ( ',' optional) ?
nativecode ::= path ( ';' path )* ( ';' parameter )*
optional   ::= '*'
```

When locating a path in a bundle the Framework must attempt to locate the path relative to the root of the bundle that contains the corresponding NativeCode clause in its manifest header or any of its fragments.

The following attributes are architected:

- osname – Name of the operating system. A number of canonical names are defined in ####
- osversion – The operating system version.
- processor – The processor architecture, see ####
- language – The ISO code for a language.
- selection-filter – A filter expression that indicates if the entry should be included or not.

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 ;
      selection-filter=
         "(org.osgi.framework.windowing.system=win32)";
      language = en ;
      language = se ,
   lib/solaris/libhttp.so ;
      osname = Solaris ;
      osname = SunOS ;
      processor = sparc,
      lib/linux/libhttp.so ;
      osname = Linux ;
      processor = mips;
      selection-filter
         = "(org.osgi.framework.windowing.system = gtk)"
```

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

If a Bundle-NativeCode clause contains duplicate 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 has probably
// not the intended effect!
```

```
Bundle-NativeCode: lib/http.DLL ;
   osname = Windows95 ;
   osversion = 3.1 ;
   osname = WindowsXP ;
   osversion = 5.1 ;
   processor = x86
```

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 in two clauses:

```
Bundle-NativeCode: lib/http.DLL ;
      osname = Windows95 ;
      osversion = 3.1;
      processor = x86,
   lib/http.DLL ;
      osname = WindowsXP ;
      osversion = 5.1;
      processor = x86
```

If the optional '*' is specified at the end of the Bundle-NativeCode manifest header, a bundle installation error will not occur if the Bundle-NativeCode header has no matching clauses.

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

```
Bundle-NativeCode: lib/win32/winxp/optimized.dll ;
      lib/win32/native.dll ;
      osname = WindowsXP ;
      processor = x86 ,
   lib/win32/native.dll ;
      osname = Windows95 ;
      osname = Windows98 ;
      osname = WindowsNT ;
      osname = Windows2000;
      processor = x86 ,
      *
```

3.10.1 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.

Certain properties can be aliased. In those cases, the manifest header should contain the generic name for that property but the Framework should attempt to include aliases when it matches. (See ####). 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 because Java 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 range includes [org.osgi.framework.os.version] or osversion is not specified
   - language ~= [org.osgi.framework.language] or language is not specified
   - selection-filter evaluates to true when using the values of the system properties or selection-filter is not specified
2 If no native clauses were selected in step 1, this algoritm is terminated and a BundleException is thrown if the optional clause is not present.
3 The selected clauses are now sorted in the following priority order:
   - osversion: floor of the osversion range in descending order
   - 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 native code library in a selected native code clause cannot be found within the bundle or any of its fragments, then the bundle installation must fail with a BundleException. This is true even if the optional clause is specified.

### Fragments are searched?? This feels very messy

If a selection filter is evaluated and its syntax is invalid, then the bundle installation must fail with a BundleException. If a selection filter is not evaluated (it may be in a native code clause where the osname or processor do not match) then the invalid filter must not cause the bundle installation to fail. This is also true even if the optional clause is specified.

### Why do we have this inconsistent behavior? This is not trivial to test because it prohibits the use of the filter class, you have to write your own filter? This the kind of stuff that makes specs complicated ...

Designing a bundle native code header can become complicated quickly when different operating systems, libraries and language are used. The best practice to design the header is to place all parameters in a table. Every targeted environment is then a row in than table. Each column is one of osname, osversion, processor and language,

OS Name	Version	Processor	Language	Filter	Libs

The previous table makes it easier to detect missing combinations. This table is then mapped to the Bundle-NativeCode header in the following table.

```
Bundle-NativeCodenativecodewin32.dll;
    delta.dll;
    processor=x86;
    osname=win32;
```

Libs	osname	osversion	processor	language	filter
nativecodewin32.dll, delta.dll	win32		x86	en	
nativecodegtk.so	linux		x86	en	(org.osgi.framework.windowing.systems=gtk)
nativecodeqt.so	linux		x86	en	(org.osgi.framework.windowing.system=qt)

Table 5 *Native code table*

```
        language=en,
    nativecodegtk.so;
        processor=x86;
        osname=linux;
        selection-filter=
            "(org.osgi.framework.windowing.system = gtk)";
        language=en,
    nativecodeqt.so;
        processor=x86;
        osname=linux;
        selection-filter =
            "(org.osgi.framework.windowing.system = qt)";
        language=en
```

If the native code algorithm fails to select a native code clause from the Bundle-NativeCode header, the bundle must fail to install and a BundleException will be thrown.

3.10.2 Runtime loading of native libraries

When a class located in a bundle attempts to load a native library the classloader of the class must first search for the native code library in the bundle associated with the classloader. If that bundle does not contain the library, then the class loader must search for the native library in the attached fragment bundles. Fragment bundles must be searched in ascending bundle id order. If the native library is not found then null must be returned to allow the parent classloader to search for the native library.

### Isnt this contradictory with the fact that the install should fail?

3.10.3 Considerations Using Native Libraries

There are some restrictions on loading native libraries due to the nature of classloaders[###7]. In order to preserve namespace separation in classloaders, only one classloader can load a native library as specified by an absolute path. Loading of a native library file by multiple classloaders (from multiple bundles, for example) will result in a linkage error.

If a fragment is attached to multiple host bundles, the fragment's native code must be accessible through a unique file for each host bundle. This is to prevent a linkage error resulting from loading of the same native library file by multiple classloaders (bundles).

A native library is unloaded only when the classloader that loaded it has been garbage collected. When a bundle is uninstalled or updated, any native libraries loaded by the bundle will not be unloaded until the bundle's classloader is garbage collected. The garbage collection will not happen until all references to objects in the bundle have been garbage collected and all bundles importing packages from the updated or uninstalled bundle are refreshed. This implies that Native Libraries loaded from the system classloader are never unloaded because the system classloader is never garbage collected.

3.10.4 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 6 on page 73, 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 6 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.

3.11 Extension Bundles

Extension bundles can deliver optional parts of the framework or boot classpath. Extensions can be used to provide functionality that must reside on the boot classpath, e.g. an implementation of java.sql, or that must be available before the Framework is running, e.g. implementation parts of the Framework.

Boot classpath extension bundles are necessary when packages in the java

### Why do we need to extend the system class path? I know I asked it but forgot. WHy do we need both?

An extension bundle must be treated as a fragment bundle that is hosted by a *magic bundle*. Magic bundles are not represented in the life cycle layer, they only exist as name. There are two magic bundle names specified:

### Is this true? Who is the host from an API point of view? It seems much more logical to have them associate with the system bundle and then have a specific header for indicating where it should be placed.

- org.osgi.framework – The symbolic name for the framework implementation bundle. Fragments of the org.osgi.framework are framework implementation extension bundles.
- org.osgi.bootclasspath – The symbolic name for the boot classpath. Fragments of the org.osgi.bootclasspath bundle are boot classpath extension bundles.

The following example uses the Fragment-Host manifest header to specify a framework implementation extension bundle.

```
Fragment-Host: org.osgi.framework
```

The following example uses the Fragment-Host manifest header to specify a bootclasspath extension bundle.

```
Fragment-Host: org.osgi.bootclasspath
```

An extension bundle, as any fragement, does not have a bundle classloader associated with it. The classes and resources delivered by an extension bundle are loaded by the boot classloader (boot classpath extension bundles) or by the framework implementation classloader (framework extension bundles).

Unlike normal fragments, extension bundle resources can not be retrieved using the Bundle object methods.

### Is there a particular reason for this?

3.11.1 Illegal ManifestHeaders for Extension Bundles

An extension bundle must throw a BundleException if it is installed or updated and it specifies any of the following headers.

- Bundle-Classpath
- Import-Package
- Require-Bundle
- ReExport-Package
- Bundle-NativeCode
- Bundle-DynamicImport
- Bundle-Activator

A framework extension bundle may specify an Export-Package header. Any exported packages specified by a framework extension bundle must be exported by the System Bundle when the Framework is re-launched.

3.11.2 Extension Bundle Lifecycle

An extension bundle lifecycle behaves the same as normal bundles installed in the framework, except that it can never be resolved while the system bundle is active. Extension bundles must be resolved before the Framework starts any bundles. The only way the extension bundle can be refreshed is by restarting the Framework.

Refreshing the extension bundle must result in a framework restart because the fragments are hosted by the system bundle.

### Does this require any external support? I think so, the boot class path can only be set at startup?

3.11.3 Classpath treatment

A boot classpath extension bundle's JAR file must be appended to the boot classpath of the host VM. A framework extension bundle's JAR is appended appended the classpath of the Framework.

Extension bundle's must be appended to their classpath in the order in which the extension bundles are installed: That is ascending bundle id order.

How a framework configures itself or the boot classpath to append the extension bundle's JAR is implementation specific. In some execution environments it may be impossible to support extension bundles. In such environments the framework must throw a BundleException when such an extension bundle is installed. The resulting Bundle Exception must have a nested exception of type UnsupportedOperationException.

3.11.4 Extension Bundles Unsupported

In an environment that has Java 2 security enabled the framework must perform an additional security check before allowing an extension bundle to be installed. In order for an extension bundle to successfully install the framework must check that the bundle location for the extension bundle has AllPermissions assigned to it (or bundle signer when using RFC 73 permissions). This means that the permissions of an extension bundle must be setup before the extension bundle is installed.

The reason AllPermission must be granted to extension bundles is because they will be loaded under the ProtectionDomain of either the boot classpath or the framework implementation. Both of these ProtectionDomains have AllPermissions granted to them. It should not be allowed for an extension bundle to be installed unless it already has been granted AllPermissions.

3.12 Localization

A bundle contains a significant amount of information that can be human readable. Some if this information may require different translations depending on the user's language, country and any special variant preferences, a.k.a. the *locale*. This section describes how a bundle can provide common translations for the manifest and other configuration resources depending on a locale.

Bundle localization resources shares a common base name. To find a potential localization resource, an underscore ('_' \###) is added plus a number of suffixes, separated by another underscore, and finally appended with the suffix .properties . The suffixes are defined in java.util.Locale. The order for the suffixes this must be:

- language
- country
- variant

For example, the following files provide manifest translations for English, Dutch (Belgium and the Netherlands) and Swedish.

```
META-INF/bundle_en.properties
META-INF/bundle_nl_BE.properties
META-INF/bundle_nl_NL.properties
META-INF/bundle_sv.properties
```

The Framework searches for localization resources by appending suffixes to the localization base name according to a specified locale and finally appending the .properties suffix. If a translation is not found, the locale must be made more generic by first removing the variant, then the country and finally the language until a resource is found that contains a valid translation. For example, looking up a translation for the locale en_GB_welsh will search in the following order:

```
META-INF/bundle_en_GB_welsh.properties
META-INF/bundle_en_GB.properties
META-INF/bundle_en.properties
META-INF/bundle.properties
```

This allows localization files for more specific locales to override localizations from less specific localization files.

3.12.1 Finding Localization Resources

Localization resources can be contained in the bundle or delivered as fragments. The Framework must therefore first look in the bundle and then in its fragments. Fragment bundles must delegate the search for a localization resource to their host bundle with the lowest bundle id.

The localization resources must be loaded only from the bundle itself or one of it's fragments irrespective of classloaders, the search must ignore any directives. The bundle will still be searched for localization files even if dot ('.') is not in the bundle classpath.

### Can we get rid of this fragment exceptions? I would like to refer to getEntry() and hten getEntry should be defined for a fragment.

3.12.2 Manifest Localization

Localized values are stored in property resources within the bundle. The default base name of the bundle localization property files is META-INF/bundle. The Bundle-Localization manifest header may optionally define another base name for the localization files. This location is relative to the root of the bundle and bundle fragments.

### why?All these options make things so complicated and seem to add very little

A localization resource contains key/value entries for localized information. All header's in a bundle's manifest that have no semantics to the Framework can be localized.

### I am not sure we should forbid localizing? The framework MUST use non localized values so will just get an error because it can't handle the '%' sign ... I think we should make it a should and not mention any headers.

A localization key can be specified as the value of a bundle's manifest header using the following syntax:

```
header-value ::= ( '%' )? text
text ::= < allowed as manifest header and property key >
```

For example, consider the following bundle manifest entries:

```
Bundle-Name: %acme bundle
Bundle-Vendor: %acme corporation
Bundle-Description: %acme description
Bundle-Activator: com.acme.bundle.Activator
Acme-Defined-Header: %acme special header
```

User defined headers can also be localized. Spaces in the localization keys are explicitly allowed.

The previous example manifest entries could be localized by the following entries in the manifest localization resource META-INF/bundle.properties.

```
# bundle.properties
acme bundle=The ACME Bundle
acme corporation=The ACME Corporation
acme description=The ACME Bundle provides all of the ACME \
services
acme special header=User Defined Acme Data
```

The above manifest entries could also have French localizations in the manifest localization file META-INF/bundle_fr_FR.properties.

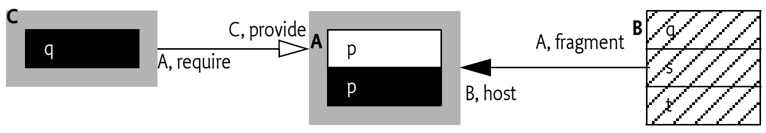
3.13 Security

3.13.1 Bundle Permission

Most package sharing permissions are based on Package Permission. However, fragments and required bundles use the bundle symbolic name to handle sharing. The Bundle Permission is used to control this type of package sharing.

The name parameter of the BundlePermission will be a bundle symbolic name. The symbolic name is used as the identifier for the *target bundle*. E.g. if a fragment A attaches to host B then A requires BundlePermission("B", "fragment") so that A is permitted to target host bundle B. The direction of the actions is depicted in Figure 32.

Figure 32 Permissions and bundle sharing



### Can target contain wildcards?

The following actions are architected:

- provide – Permission to provide packages to the target bundle.
- require – Permission to require packages from the target bundle.
- host – Permission to attach to the target fragment bundle.
- fragment – Permission to attach as a fragment to the target host bundle.

### I have this uneasy feeling that permission based on symbolic names are going to be ared herring ... we should seriously consider this permission

3.13.2 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.

### If we would add a RESOURCE permission we could probably skip the AdminPermission "resource", then this would be in the right layer. E.g. the http server would get PackagePermission(*,RESOURCE). Idea?

3.14 Class Loader Changes Since R3

Many!!

3.15 References

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

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

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

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

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

[25] *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

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

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

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

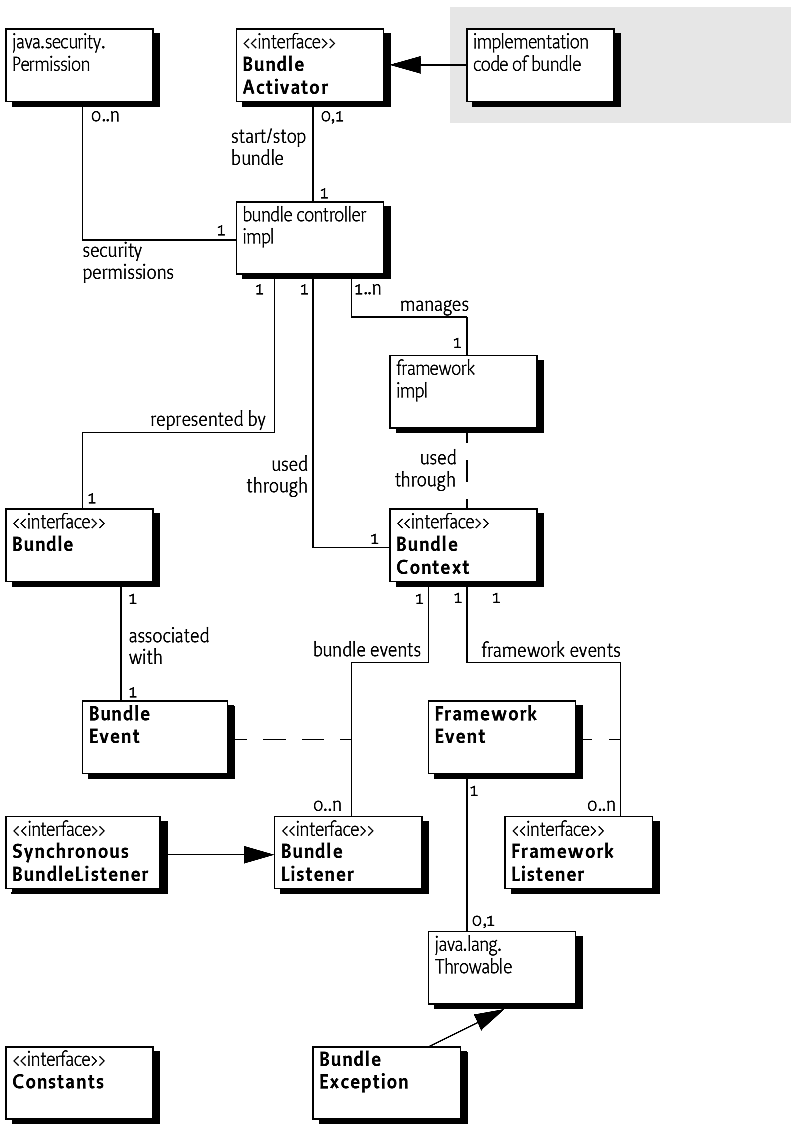
[29] *W3C EBNF*
http://www.w3c.org/TR/REC-xml#sec-notation

[30] *Lexical Structure Java Laguage*
http://java.sun.com/docs/books/jls/second_edition/html/lexical.doc.html

[31] *Interval Notation*
http://www.math.ohio-state.edu/courses/math104/interval.pdf

4 Life Cycle Layer

Figure 33 *Class Diagram* org.osgi.framework



4.0.1 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 97 for more information about the starting and stopping of the Framework.
- The system bundle's `Bundle.getHeaders` method returns a `Dictionary` object with implementation-specific manifest headers. For example, the system bundle's manifest file should contain an Export-Package header declaring which packages are to be exported by the Framework (for example, `org.osgi.framework`).

4.1 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.

- 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.

4.1.1 Bundle Identifiers

A bundle is identified by a number of names that vary in their scope:

- *Bundle Id* – A `long` that is a Framework assigned unique identifier for the full life time of a bundle, even if it is updated or the Framework is restarted. Its purpose is to distinguish bundles in a Framework. Bundle ids are assigned in ascending order to bundles when they are installed. The method `getBundleId()` returns a bundle's identifier.
- *Bundle location* – A name assigned by the management agent (Operator) to a bundle during the installation. This string is normally interpreted as a URL to the JAR file but this is not mandatory. Within an instance of a Framework, a location must be unique. A location string uniquely iden-

tifies a bundle and must not change when a bundle is updated. The getLocation() method retrieves the location of a bundle.

- *Bundle Symbolic Name* – A name assigned by the developer. The combination of Bundle Version and Bundle Symbolic Name is a globally unique identifier for a bundle. The getSymbolicName() method returns the assigned bundle name.

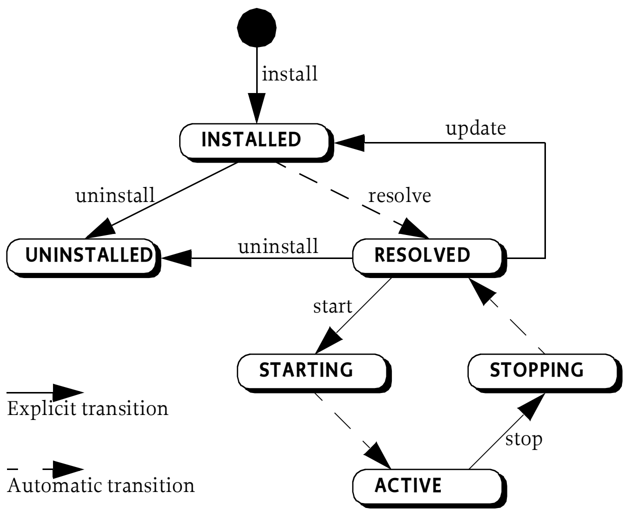
### what happens with a Bundle Symbolic name in an update when the developer screws up and asssigns a new symbolic name? I guess this should be rejected?

4.1.2 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 34 *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 187.

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

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

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

4.1.3 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 67.

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

4.1.4 Resolving Bundles

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

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

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

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

4.1.5 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 187.

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. However, the bundle must still be persistently marked as startable. A bundle marked as startable and whose start level is appropriate must be automatically started as soon as the bundle becomes resolveable, also if this happens after a Framework restart.

### Is this only for unresolved or also if the activator throws a method?

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 unnec-

essary to unregister services or Framework listeners because they must be cleaned up by the Framework anyway.

4.1.6 Stopping Bundles

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

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

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

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

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

4.1.7 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 [35] *The Java Language Specification*, Section 13.5) with its counterpart in Older.
- For any package exported by Older, Newer must export the same package, which must be compatible with its counterpart in Older.

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

4.1.8 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.

4.1.9 Detecting Bundle Changes

The Bundle object provides a convenient way to detect changes in a bundle. The Framework must maintain a last modified time that is changed for any of the life cycle operations. The getLastModified() method will return the last time the bundle was installed, updated, or uninstalled.

What about the attachment of a fragment?

The method must return the number of milliseconds since midnight Jan 1, 1970 UTC with the exception that a change must always result in a higher value than the previous last modified time.

The getLastModified() is very useful when a bundle is caching resources from another bundle. For example, translations or configuration resources. This caching bundle must then update its cached information when the target bundle is updated or uninstalled. The life cycle change of the target bundle can happen while the caching bundle is not active. The last modified time is therefore a convenient way to track these target bundles.

### yes?

4.1.10 Retrieving Manifest Headers

The Bundle interface defines two method to return manifest header information: getHeaders() and getHeaders(String).

- getHeaders() – Returns a Dictionary object that contains the bundle's manifest headers and values as key/value pairs. The values returned are localized according to the default locale returned by java.util.Locale.getDefault.

- getHeaders(String) – Returns a Dictionary object that contains the bundle's manifest headers and values as key/value pairs. The values returned are localized using the specified locale. The locale may take the following values:
  - null – The defalut locale returned by java.util.Locale.getDefault is used. This makes this method identical to the getHeaders() method.
  - Empty string – The dictionary will contain the raw (unlocalized) manifest headers including any leading '%'.
  - Locale – The given locale is used to localize the manifest headers.

Localization is performed according to the description in *Localization* on page 75. If no translation is found for a specific key, the Dictionary returned by Bundle.getHeaders will return the raw values as specified in the manifest header values without the leading '%'.

These methods require AdminPermission because some of the manifest header information may be sensitive, such as the packages listed in the Export-Package header.

The getHeaders methods must continue to provide the manifest header information after the bundle enters the INSTALLED state. After the bundle has been uninstalled, this method will only return manifest headers that are raw or localized for the default locale at the time the bundle was uninstalled.

A framework implementation must use only the raw (unlocalized) manifest headers when processing manifest headers. I.e. localizations must not influence the operations on a bundle that the framework performs.

4.1.11 Access To Resources and Classes

The JAR file of a bundle contains classes and other resources. Normally, the class space of a bundle is used to access both the classes and the resources. A class space ensures that the myriad of class loading rules are applied and that the class space is consistent. I.e. the actual resource loaded may come from another bundle than the bundle it was asked from.

The loadClass(String), getResource(String) and getResources(String) provide access to resources that is consistent with the class space, with the.exception when the bundle is not, or can not be, resolved, in that case only the local bundle is searched.

### how are split packages treated for getResources()?

- getResource(String) – Return a URL to a resource taking classloading rules into account
- getResources(String) – Return an Enumeration of URL objects to resources.

### How is this handled when there a fragments with overlapping packages?

- loadClass(String) – Return a class object for the given class.

The aforementioned calls must preserve the class loading rules as described in *Overall Search Order* on page 62. I.e. they must follow the bundle class path, imports and exports, as well as fragments. If the bundle is unresolved, then the Framework must try to resolve the bundle first. If this fails, the Framework must still obey the Bundle-Classpath but must ignore any potential fragments.

Unfortunately, the `loadClass`, `getResource` and `getResources` methods require (or could even create) a classloader. Class loader are expensive resources. Therefore, the Framework provides the following methods to access bundle's JAR file without creating or requireing a classloader:

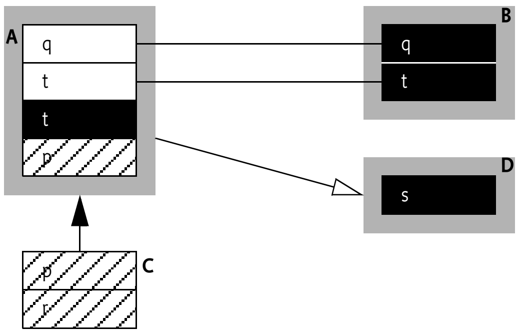
- `getEntry(String)` – The `getEntry` method accesses the raw JAR file without requiring a classloader.
- `getEntryPaths(String)` – The `getEntryPaths` method returns an Enumeration of `String` objects, representing the paths to resources in the given package.

For example, consider the following setup:

```
A: Require-Bundle: D
   Import-Package: q,t
   Export-Package: t
   Private-Package: p
B: Export-Package: q,t
C: Fragment-Host: A
   Private-Package: p,r
D: Export-Package: s
```

This setup is depicted in Figure 35.

Figure 35 *Setup for showing the difference between getResource and getEntry*



The following table shows the effect of getting a resource from this setup.

Resource	**getResource**	**loadClass**	**getEntry**
q	B.q	B.q	A.q
p	C.p + A.p	C.p+A.p	A.p
r	C.r	C.r	null

Table 7 *Differences between getResource, loadClass and getEntry*

Resource	getResource	loadClass	getEntry
s	D.s	D.s	null
t	B.t	B.t	A.t
bundle not resolveable			
q	null	null	A.q
...	null	null	

Table 7 *Differences between getResource, loadClass and getEntry*

### nice inconsistency in naming! getResource/getResources and getEntry and getEntry(Path)s, also return Enumeration is different.

4.1.12 Permissions of a Bundle

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

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

4.2 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 84.
- Interrogating other bundles installed in the OSGi environment. See *Getting Bundle Information* on page 91.
- Obtaining a persistent storage area. See *Persistent Storage* on page 91.
- Retrieving service objects of registered services. See *ServiceReference Objects* on page 162.
- Registering services in the Framework service. See *Registering Services* on page 163.
- Subscribing or unsubscribing to events broadcast by the Framework. See *Listeners* on page 94.

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

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

4.2.1 Getting Bundle Information

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

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

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

4.2.2 Persistent Storage

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

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

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

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

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

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

4.2.3 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 [39] *Codes for the Representation of Names of Languages* for valid values.
org.osgi.framework.executionenvironment	A comma separated list of provided Execution Environments (EE). All methods of each listed EE must be present on the Service Platform. For example, this property could contain: `CDC-1.0/Foundation-1.0,OSGi/Minimum-1.0` A Service Platform implementation must provide *all* the signatures that are defined in the mentioned EEs. Thus the Execution Environment for a specific Service Platform Server must be the combined set of all signatures of all EEs in the org.osgi.framework.executionenvironment property.
org.osgi.framework.processor	Processor name. The following table defines a list of processor names. New processors are made available on the OSGi web site in the Developers Zone. Names should be matched case insensitive.

Name	Aliases	Description
68k		68000 and up
ARM		Intel Strong ARM
Alpha		Compaq
Ignite	psc1k	PTSC
Mips		SGI
PArisc		Hewlett Packard
PowerPC	power ppc	Motorola/IBM
Sparc		SUN

Table 8 Property Names

Property name	Description
	x86 — pentium i386 i486 i586 i686 — Intel
	x86-64 — amd64 — New 64 bit x86 architecture
org.osgi.framework.os.version	The version of the operating system. If the version does not fit the standard x.y.z format (e.g. 2.4.32-kwt), then the Operator should define a System property with this name.
org.osgi.framework.os.name	The name of the operating system (OS) of the host computer. The following table defines a list of OS names. New OS names are made available on the OSGi web site in the Developers Zone. Names should be matched case insensitive.

Name	Aliases	Description
AIX		IBM
DigitalUnix		Compaq
FreeBSD		Free BSD
HPUX		Hewlett Packard
IRIX		Silicon Graphics
Linux		Open source
MacOS		Apple
Netware		Novell
OpenBSD		Open source
NetBSD		Open source
OS2	OS/2	IBM
QNX	procnto	QNX
Solaris		Sun Micro Systems
SunOS		Sun Micro Systems
VxWorks		WindRiver Systems
Win32	Win*	
Windows95	Win95 Windows 95	Microsoft Windows 95
Windows98	Win98 Windows 98	Microsoft Windows 98
WindowsNT	WinNT Windows NT	Microsoft Windows NT

Table 8 *Property Names*

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

Table 8 *Property Names*

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

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

4.3 Events

The OSGi Framework Life Cycle layer supports the following types of events:

- 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.

The actual event that is reported is available with the getType method. The integer that is returned from this method can be one of the constant names that is described in the class. However, events can, and will be, extended in the future. Unrecognized event types should be ignored.

4.3.1 Listeners

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

- BundleListener and SynchronousBundleListener – Called with an event of type BundleEvent when a bundle's life cycle information has been changed.
  SynchronousBundleListener objects are called synchronously during the processing of the event, and must be called before any BundleListener object is called. The following events are send by the Framework after it has moved to a different state.

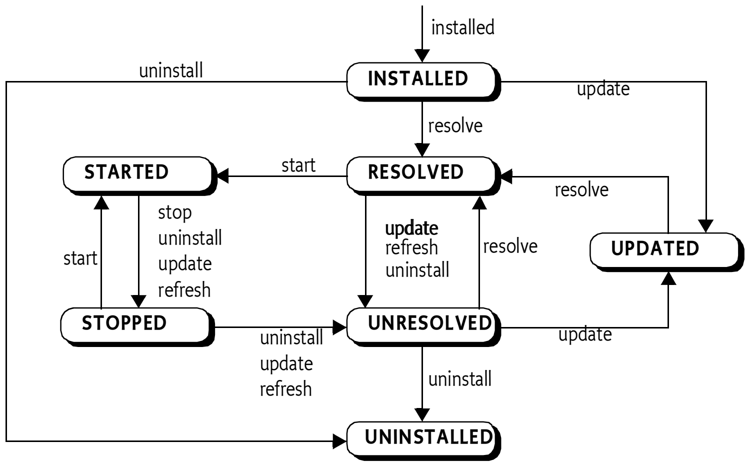
### I realized here that we do not really send state changes but that have some "shadow state". The UNRESOLVED event is kind of troublesome because we do not have a corresponding state.Note sure what to do, it almost looks like we need an unresolved state ... Or do we need another listener that follows the state machine? This is much more messy than I expected.

- INSTALLED – Send after a bundle is installed.
- RESOLVED – Send when the Framework has resolved a bundle.
- STARTED – Send when the Framework has started a bundle.
- STOPPED – Send when the Framework has stopped a bundle.
- UNINSTALLED – Send when the Framework has uninstalled a bundle
- UNRESOLVED – Send when the Framework detects that a bundle becomes unresolved, this could happen when the bundle is uninstalled, updated or refreshed. When a set of bundles are refreshed using the PackageAdmin API then each bundle in the set must have an UNRESOLVED BundleEvent published. The UNRESOLVED BundleEvent must be published after all the bundles in the set have been stopped and, in the case of a synchronous bundle listener, *before* any of the bundles in the set are re-started.
- UPDATED – Published after a bundle is updated.

Figure 36 shows the relations between the events. The diagram assumes a number of pseudo states (these differ from the defined Bundle states). These pseudo states are assumed to generate an event when they are entered by the arrow. The ellipses contain the name of the event that is sent out after a process shown on the arrow. E.g., the install process moves the state to the INSTALLED pseudo states, this generates the INSTALLED event. If the started event is received, the uninstall process moves from to the pseudo state STOPPED, to UNRESOLVED to UNINSTALLED.

### This diagram needs good checking, it is tricky

Figure 36 Event sequence



- FrameworkListener – Called with an event of type FrameworkEvent. Framework events are of type:

- ERROR – Important error that requires the immediate attention of an operator.
- INFO – General information that is of interest in special situations.
- PACKAGES_REFRESHED – The Framework has refreshed the packages.
- STARTED – The Framework has performed all initializations and is running in normal mode.
- STARTLEVEL_CHANGED – Is send by the Framework after a new start level has been set and processed.
- WARNING – A warning to the operator which is not crucial but may indicate a potential error situation.

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

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).

4.3.2 Delivering Events

When delivering an event asynchronously, the Framework must:

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

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

The following three scenarios illustrate this concept.

1. Scenario 1 event sequence:
   - Event A is published.
   - Listener 1 is registered.
   - Asynchronous delivery of Event A is attempted.

   Expected Behavior: Listener 1 must not receive Event A, because it was not registered at the time the event was published.
2. Scenario 2 event sequence:
   - Listener 2 is registered.
   - Event B is published.
   - Listener 2 is unregistered.
   - Asynchronous delivery of Event B is attempted.

   Expected Behavior: Listener 2 receives Event B, because Listener 2 was registered at the time Event B was published.

3. Scenario 3 event sequence:
   - Listener 3 is registered.
   - Event C is published.
   - The bundle that registered Listener 3 is stopped.
   - Asynchronous delivery of Event C is attempted.

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

4.3.3 Synchronization Pitfalls

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

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

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

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

4.4 Framework Startup and Shutdown

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

4.4.1 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 94.
2. The system bundle enters the `STARTING` state. More information about the system bundle can be found in *The System Bundle* on page 81.
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 82. If the Framework implements the optional Start Level specification, this behavior is different. See *Start Level Service Specification* on page 187.

4. The system bundle enters the ACTIVE state.
5. A Framework event of type FrameworkEvent.STARTED is broadcast.

4.4.2 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 81. 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 187.
3. Event handling is disabled.

4.5 The Framework on Java 1.1

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

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

4.5.1 ClassLoader.getResource

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

4.5.2 ClassLoader.findLibrary

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

4.5.3 Resource URL

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

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

4.5.4 Comparable

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

4.6 Security

4.6.1 AdminPermission

The Admin Permission is a permission used to grant the right to manage the Framework with the option to restrict the right to manage only a subset of bundles, the so called *targets*. For example, a bundle can be given the right to only manage bundles of a signer that has a subject name of ACME:

```
org.osgi.framework.AdminPermission("(subject.cn=ACME)",
   ... )
```

The actions of the Admin Permission are fine grained. They allow the deployer to assign only the permissions that are necessary for a bundle. For example, an Http implementation could be granted access to all resources of all bundles.

```
org.osgi.framework.AdminPermission("*",
   "resource" )
```

### Do we support *

Code that needs to check Admin Permission must always use the constructor that takes a bundle id (long) as parameter: AdminPermission(String, String) ### with a single action. This is to ensure the fastest execution of the permission check. If no target is implied, the id must be 0 to refer to the system bundle.

For example, the implementation of Bundle.loadClass must check that the caller has access to its class space:

```
public class BundleImpl implements Bundle {
   long      id;
   ...
```

```
        Class loadClass(String name) {
           SecurityManager.checkPermission(
              new AdminPermission(id,"class") );
              ...
        }
     }
```

4.6.1.1 Target filters

The external form of Permission Admin can take a number of different descriptions for the targets.because the bundle id is not known a priori. In the external form, if the permission is intended to address all possible targets, the '*' can be used.

Otherwise, a filter expression can be used to select a set of targets. The following keys can be used in the filter expression:

- id – A long specifying the id of the target bundle.
- location – The location of the target.
- subject.<dn aspect> – Each bundle can be associated with one or more certificates. Each certificate has a subject and an issuer X.500 Distinguished Name (a.k.a. DN). The different fields of the DN can be tested in the expression. The following DN aspects can be tested for:
  - cn – Common Name
  - l – Location
  - st – State
  - o – Organisation name.
  - ou – Organizational Unit
  - c – Country
  - street – Street address
  - dc – Domain Component (###???)
  - uid – User ID
- issuer.<dn aspect> – The different fields of the issuer's DN. The DN aspects are identical.
- bundle.symbolic.name – The symbolic name of a bundle
- bundle.version – The version of the bundle

### Added version and changed from name to bundle-symbolic name. BTW, it is kind of weird to have bundle-symbolic-name and bundle.symbolic.name, maybe we should standardize?

Can we have an attribute that is a domain name and the allow the use of SSL certificates? The DN is kind of cumbersome

An example that selects all bundles with a certificate signed by the ACME corporation but using the certificate from the Netherands group.

```
(&(subject.cn=ACME)(issuer.cn=Thawte))
```

### verify

Another example is to verify for the bundle symbolic name by treating them as a reverse domain name.

```
(bundle.symbolic.name=com.acme.*)
```

This example requires strict control of the permission for a bundle to use a symbolic name.

4.6.1.2 Actions

The action parameter of AdminPermission will specify the subset of privileged administrative operations thar are allowed by the Permission Admin. The actions that are architected are listed in table###. Future version of the spec, as well as additional system services, can add additional actions. The given set should therefore not be assumed a closed set.

Action	Used in
`metadata`	`Bundle.getHeaders` `Bundle.getLocation`

`resource`	`Bundle.getResource`
	`Bundle.getEntry`
	`Bundle.getEntryPaths`
	`Bundle resource/entry URL creation`

`class`	`Bundle.loadClass`

`lifecycle`	`BundleContext.installBundle`
	`Bundle.update`
	`Bundle.uninstall`
	`StartLevel.setBundleStartLevel`

`execute`	`Bundle.start` `Bundle.stop`

`listener`	`BundleContext.addBundleListener for SynchronousBundleListener`
	`BundleContext.removeBundleListener for SynchronousBundleListener`

`permission`	`PermissionAdmin.setPermissions` `PermissionAdmin.setDefaultPermissions`
`resolve`	`PackageAdmin.refreshPackages` `PackageAdmin.resolveBundles`
`startlevel`	`StartLevel.setStartLevel` `StartLevel.setInitialBundleStartLevel`

Table 9

The special action "*" will represent all actions.

Each bundle must be given AdminPermission(<bundle id>, "resource") so that it can access it's own resources. This is an implicit permission that must be given to all bundles. This replaces the framework implementation defined permission in.

### This kind of unpleasant, this is an module layer issue but AdminPermission is mainly a life cycle layer issue ...

4.7 Life Cycle Layer Changes since R3

4.7.1 AdminPermission

To provide backwards compatibility for existing code, we define the no argument constructor for AdminPermission to be the same as constructing AdminPermission with a name argument of "*" and an actions argument of "*". This results in an AdminPermission which applies to all bundle id and all actions. The no argument constructor will be deprecated to indicate that the 2 argument constructor should now be used.

4.8 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;version=1.3
```

4.8.1 Summary

- *AdminPermission* - Indicates the caller's authority to perform lifecycle operations on or to get sensitive information about a bundle. [p.103]
- *AllServiceListener* - A `ServiceEvent` listener. [p.103]
- *Bundle* - An installed bundle in the Framework. [p.104]
- *BundleActivator* - Customizes the starting and stopping of this bundle. [p.115]
- *BundleContext* - A bundle's execution context within the Framework. [p.116]
- *BundleEvent* - A Framework event describing a bundle lifecycle change. [p.126]
- *BundleException* - A Framework exception used to indicate that a bundle lifecycle problem occurred. [p.82]
- *BundleListener* - A BundleEvent listener. [p.128]
- *BundlePermission* - A bundle's authority to require or provide or specify a host bundle. [p.129]
- *Configurable* - Supports a configuration object. [p.130]
- *Constants* - Defines standard names for the OSGi environment property, service property, and Manifest header attribute keys. [p.131]
- *Filter* - An RFC 1960-based Filter. [p.143]
- *FrameworkEvent* - A general Framework event. [p.144]
- *FrameworkListener* - A FrameworkEvent listener. [p.146]
- *InvalidSyntaxException* - A Framework exception. [p.147]
- *PackagePermission* - A bundle's authority to import or export a package. [p.148]
- *ServiceEvent* - A service lifecycle change event. [p.149]
- *ServiceFactory* - Allows services to provide customized service objects in the OSGi environment. [p.150]
- *ServiceListener* - A ServiceEvent listener. [p.151]

- *ServicePermission* - Indicates a bundle's authority to register or get a service. [p.152]
- *ServiceReference* - A reference to a service. [p.153]
- *ServiceRegistration* - A registered service. [p.155]
- *SynchronousBundleListener* - A synchronous BundleEvent listener. [p.156]
- *Version* - Version identifier for bundles and packages. [p.157]

4.8.2 public final class AdminPermission extends BasicPermission

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

### needs to be updated for RFC 73! AdminPermission has no actions or target.

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

4.8.2.1 public AdminPermission( )

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

4.8.2.2 public AdminPermission(String name, String actions)

name Ignored; always set to "AdminPermission".

actions Ignored.

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

4.8.2.3 public boolean equals(Object obj)

obj The object being compared for equality with this object.

□ Determines the equality of two AdminPermission objects.

Two AdminPermission objects are always equal.

Returns true if obj is an AdminPermission; false otherwise.

4.8.2.4 public boolean implies(Permission p)

p The permission to interrogate.

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

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

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

4.8.2.5 public PermissionCollection newPermissionCollection( )

□ Returns a new PermissionCollection object suitable for storing AdminPermissions.

Returns A new PermissionCollection object.

4.8.3 public interface AllServiceListener extends ServiceListener

A ServiceEvent listener.

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

An AllServiceListener object is registered with the Framework using the BundleContext.addServiceListener method. AllServiceListener 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 AllServiceListener 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.

Unlike normal ServiceListener objects, AllServiceListener objects have all ServiceEvent objects delivered regardless of the package scope of the listening bundle.

See Also ServiceEvent[p.149] , ServicePermission[p.152]

4.8.4 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.106]
- INSTALLED[p.105]
- RESOLVED[p.105]
- STARTING[p.105]
- STOPPING[p.105]
- ACTIVE[p.105]

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.

4.8.4.1 public static final int ACTIVE = 32

This bundle is now running.

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

The value of ACTIVE is 0x00000020.

4.8.4.2 public static final int INSTALLED = 2

This bundle is installed but not yet resolved.

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

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

The value of INSTALLED is 0x00000002.

4.8.4.3 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.131] Manifest header.
- The bundle's package dependencies from its Constants.EXPORT_PACKAGE[p.135] and Constants.IMPORT_PACKAGE[p.138] Manifest headers.

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

The value of RESOLVED is 0x00000004.

4.8.4.4 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.111] method is active. A bundle will be in this state when the bundle's BundleActivator.start[p.115] is called. If this method completes without exception, then the bundle has successfully started and will move to the ACTIVE state.

The value of STARTING is 0x00000008.

4.8.4.5 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.112]` method is active. A bundle will be in this state when the bundle's `BundleActivator.stop[p.116]` method is called. When this method completes the bundle is stopped and will move to the RESOLVED state.

The value of STOPPING is 0x00000010.

4.8.4.6 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.

4.8.4.7 public long getBundleId( )

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

A bundle's unique identifier has the following attributes:

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

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

Returns The unique identifier of this bundle.

4.8.4.8 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

4.8.4.9 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 (String objects###)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

4.8.4.10 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 according to 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.132]

4.8.4.11 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 according to 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 and the raw (unlocalized) header values, including any leading "%", will be returned.

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.107], `Constants.BUNDLE_LOCALIZATION`[p.132]

Since 1.3

4.8.4.12 public long getLastModified( )

□ Returns the time when this bundle was last modified. A bundle is considered to be modified when it is installed, updated or uninstalled.

The time value is the number of milliseconds since January 1, 1970, 00:00:00 GMT.

Returns The time when this bundle was last modified.

Since 1.3

4.8.4.13 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.

4.8.4.14 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.155], `ServiceReference`[p.153], `ServicePermission`[p.152]

4.8.4.15 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. If this bundle is a fragment bundle then null is returned.

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

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

Since 1.1

4.8.4.16 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. If this bundle is a fragment bundle then null is returned. ### How are split packages treated??? ### shouldn't we mention that this can create a classloader? ### How is the Bundle-Class path treated

Returns an Enumeration of URLs to the named resources, or null if the resource could not be found or if this bundle is a fragment bundle 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

4.8.4.17 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.153], `ServicePermission`[p.152]

4.8.4.18 public int getState( )

□ Returns this bundle's current state.

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

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

4.8.4.19 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

4.8.4.20 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.

4.8.4.21 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.144] is broadcast containing a BundleException with details of the reason the bundle could not be resolved. This method must then throw a ClassNotFoundException.

If the bundle is a fragment bundle then this method must 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 this bundle is a fragment bundle 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

4.8.4.22 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.115] 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.126] 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.

4.8.4.23 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.116] 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.126] is broadcast.

Preconditions

- getState() in {ACTIVE}.

Postconditions, no exceptions thrown

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

Postconditions, when an exception is thrown

- None.

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

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

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

4.8.4.24 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.144] is broadcast containing the exception.
3 This bundle's state is set to UNINSTALLED.
4 A bundle event of type `BundleEvent.UNINSTALLED`[p.127] 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.112]

4.8.4.25 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.134] 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.127] 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.144] 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.112], `start()` [p.111]

4.8.4.26 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.113]

4.8.5 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: <i>class-name</i>
```

where class-name is a fully qualified Java classname.

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

4.8.5.1 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.111]

4.8.5.2 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.112]

4.8.6 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.104] 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.115] method. The same BundleContext object will be passed to this bundle when it is stopped using the `BundleActivator.stop`[p.116] 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.

4.8.6.1 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.119] 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.126] , `BundleListener`[p.128]

4.8.6.2 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.119] 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.144] , `FrameworkListener`[p.146]

4.8.6.3 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.119] for a definition of context bundle, and `Filter`[p.143] 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.139] 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.149], `ServiceListener`[p.151], `ServicePermission`[p.152]

4.8.6.4 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.117]

4.8.6.5 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.143] for a description of the filter string syntax.

If the filter cannot be parsed, an `InvalidSyntaxException`[p.147] will be thrown with a human readable message where the filter became unparsable.

Returns A Filter object encapsulating the filter string.

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

`NullPointerException` – If filter is null.

Since 1.1

4.8.6.6 public ServiceReference[] getAllServiceReferences(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.143] 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.147] 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.139] property.
4. An array of ServiceReference to the selected services is returned.

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

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

4.8.6.7 public Bundle getBundle( )

□ Returns the Bundle object for this context bundle.

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

Returns The context bundle's Bundle object.

Throws `IllegalStateException` – If this context bundle has stopped.

4.8.6.8 public Bundle getBundle(long id)

id The identifier of the bundle to retrieve.

□ Returns the bundle with the specified identifier.

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

4.8.6.9 public Bundle[] getBundles( )

□ Returns a list of all installed bundles.

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

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

4.8.6.10 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.119] for a definition of context bundle.

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

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

Throws `IllegalStateException` – If the context bundle has stopped.

4.8.6.11 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.138] - The OSGi Framework version.
- `Constants.FRAMEWORK_VENDOR`[p.138] - The Framework implementation vendor.
- `Constants.FRAMEWORK_LANGUAGE`[p.137] - The language being used. See ISO 639 for possible values.
- `Constants.FRAMEWORK_OS_NAME`[p.137] - The host computer operating system.
- `Constants.FRAMEWORK_OS_VERSION`[p.137] - The host computer operating system version number.
- `Constants.FRAMEWORK_PROCESSOR`[p.138] - 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.133] Manifest header's matching algorithm for selecting native language code.

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

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

4.8.6.12 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.120] the context bundle's use count for that service is incremented by one. Each time the service is released by `ungetService`[p.125] 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.119] 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.151] 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.144] 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.125] , `ServiceFactory`[p.150]

4.8.6.13 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.122] 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.141] 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.141] 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.122]

4.8.6.14 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.143] 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.147] 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.139] property.
4 An array of ServiceReference to the selected services is returned.

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

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

4.8.6.15 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.82] 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.126] 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.

4.8.6.16 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.122]

4.8.6.17 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.139].

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.131] 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.155] 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.119] for a definition of context bundle. Other bundles can locate the service by using either the getServiceReferences[p.122] or getServiceReference[p.121] method.

A bundle can register a service object that implements the ServiceFactory[p.150] 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.141] identifying the registration number of the service, and a property named Constants.OBJECTCLASS[p.139] 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.150] 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.155], `ServiceFactory`[p.150]

4.8.6.18 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.124] 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.139] 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.124]

4.8.6.19 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.119] for a definition of context bundle.

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

Throws `IllegalStateException` – If this context bundle has stopped.

4.8.6.20 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.119] for a definition of context bundle.

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

Throws `IllegalStateException` – If this context bundle has stopped.

4.8.6.21 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.119] for a definition of context bundle.

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

Throws `IllegalStateException` – If this context bundle has stopped.

4.8.6.22 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.119] 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.151] 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.120], `ServiceFactory`[p.150]

4.8.7 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.

4.8.7.1 public static final int INSTALLED = 1

The bundle has been installed.

The value of INSTALLED is 0x00000001.

See Also `BundleContext.installBundle(String)`[p.122]

4.8.7.2 public static final int RESOLVED = 32

The bundle has been resolved.

The value of RESOLVED is 0x00000020.

See Also `Bundle.RESOLVED`[p.105]

Since 1.3

4.8.7.3 public static final int STARTED = 2

The bundle has been started.

The value of STARTED is 0x00000002.

See Also `Bundle.start`[p.111]

4.8.7.4 public static final int STOPPED = 4

The bundle has been stopped.

The value of STOPPED is 0x00000004.

See Also `Bundle.stop`[p.112]

4.8.7.5 public static final int UNINSTALLED = 16

The bundle has been uninstalled.

The value of UNINSTALLED is 0x00000010.

See Also `Bundle.uninstall`[p.113]

4.8.7.6 public static final int UNRESOLVED = 64

The bundle has been unresolved.

The value of UNRESOLVED is 0x00000040.

See Also `Bundle.INSTALLED`[p.105]

Since 1.3

4.8.7.7 public static final int UPDATED = 8

The bundle has been updated.

The value of UPDATED is 0x00000008.

See Also `Bundle.update()`[p.113]

4.8.7.8 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.

4.8.7.9 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.

4.8.7.10 public int getType( )

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

- `INSTALLED`[p.126]
- `STARTED`[p.126]
- `STOPPED`[p.126]
- `UPDATED`[p.127]
- `UNINSTALLED`[p.127]
- `RESOLVED`[p.126]
- `UNRESOLVED`[p.127]

Returns The type of lifecycle event.

4.8.8 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.

4.8.8.1 public BundleException(String msg, Throwable cause)

msg The associated message.

cause The cause of this exception.

□ Creates a BundleException that wraps another exception.

4.8.8.2 public BundleException(String msg)

msg The message.

□ Creates a BundleException object with the specified message.

4.8.8.3 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

4.8.8.4 public Throwable getNestedException( )

□ Returns any nested exceptions included in this exception.

This method predates the general purpose exception chaining mechanism. The `getCause()`[p.128] method is now the preferred means of obtaining this information.

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

4.8.8.5 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

4.8.9 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.117] method. BundleListener s are called with a BundleEvent object when a bundle has been installed, started, stopped, updated, or uninstalled.

See Also `BundleEvent`[p.126]

4.8.9.1 public void bundleChanged(BundleEvent event)

event The BundleEvent.

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

4.8.10 public final class BundlePermission extends BasicPermission

A bundle's authority to require or provide or specify a host bundle.

A bundle symbolic name defines a unique fully qualified name.

For example:

```
org.osgi.example.bundle
```

`BundlePermission` has four actions: `PROVIDE`, `REQUIRE`, `HOST`, and `FRAGMENT`. The `PROVIDE` action implies the `REQUIRE` action.

Since 1.3

4.8.10.1 public static final String FRAGMENT = "fragment"

The action string `fragmentBundle`.

4.8.10.2 public static final String HOST = "host"

The action string `fragmentHost`.

4.8.10.3 public static final String PROVIDE = "provide"

The action string `provideBundle`.

4.8.10.4 public static final String REQUIRE = "require"

The action string `requireBundle`.

4.8.10.5 public BundlePermission(String symbolicName, String actions)

symbolicName the bundle symbolic name.

actions `PROVIDE`, `REQUIRE`, `HOST`, `FRAGMENT` (canonical order).

□ Defines the authority to provide and/or require and or specify a host fragment symbolic name within the OSGi environment.

Bundle Permissions are granted over all possible versions of a bundle. A bundle that needs to provide a bundle must have the appropriate `BundlePermission` for the symbolic name; a bundle that requires a bundle must have the appropriate `BundlePermssion` for that symbolic name; a bundle that specifies a fragment host must have the appropriate `BundlePermission` for that symbolic name.

4.8.10.6 public boolean equals(Object obj)

obj The object to test for equality with this `BundlePermission` object.

□ Determines the equality of two `BundlePermission` objects. This method checks that specified bundle has the same bundle symbolic name and `BundlePermission` actions as this `BundlePermission` object.

Returns `true` if `obj` is a `BundlePermission`, and has the same bundle symbolic name and actions as this `BundlePermission` object; `false` otherwise.

4.8.10.7 public String getActions( )

□ Returns the canonical string representation of the `BundlePermission` actions.

Always returns present `BundlePermission` actions in the following order: `PROVIDE`, `REQUIRE`, `HOST`, `FRAGMENT`.

Returns Canonical string representation of the `BundlePermission` actions.

4.8.10.8 public int hashCode( )

□ Returns the hash code value for this object.

Returns A hash code value for this object.

4.8.10.9 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 symbolic name of the target is implied by the symbolic name of this object. The list of `BundlePermission` actions must either match or allow for the list of the target object to imply the target `BundlePermission` action.

The permission to provide a bundle implies the permission to require the named symbolic name.

```
x.y.*,"provide" -> x.y.z,"provide" is true
*,"require" -> x.y, "require"      is true
*,"provide" -> x.y, "require"      is true
x.y,"provide" -> x.y.z, "provide"  is false
```

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

4.8.10.10 public PermissionCollection newPermissionCollection( )

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

Returns A new `PermissionCollection` object.

4.8.11 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

4.8.11.1 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.103] 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

4.8.12 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.107], `BundleContext.getProperty`[p.120], `BundleContext.registerService(String[],Object, Dictionary)`[p.124]

Since 1.1

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

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

If present, this header specifies the name of the bundle resource class that implements the BundleActivator interface and whose start and stop methods are called by the Framework when the bundle is started and stopped, respectively.

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

4.8.12.2 public static final String BUNDLE_CATEGORY = "Bundle-Category"

Manifest header (named "Bundle-Category") identifying the bundle's category.

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

4.8.12.3 public static final String BUNDLE_CLASSPATH = "Bundle-ClassPath"

Manifest header (named "Bundle-ClassPath") identifying a list of nested JAR files, which are bundle resources used to extend the bundle's classpath.

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

4.8.12.4 public static final String BUNDLE_CONTACTADDRESS = "Bundle-ContactAddress"

Manifest header (named "Bundle-ContactAddress") identifying the contact address where problems with the bundle may be reported; for example, an email address.

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

4.8.12.5 public static final String BUNDLE_COPYRIGHT = "Bundle-Copyright"

Manifest header (named "Bundle-Copyright") identifying the bundle's copyright information, which may be retrieved from the Dictionary object returned by the Bundle.getHeaders method.

4.8.12.6 public static final String BUNDLE_DESCRIPTION = "Bundle-Description"

Manifest header (named "Bundle-Description") containing a brief description of the bundle's functionality.

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

4.8.12.7 public static final String BUNDLE_DOCURL = "Bundle-DocURL"

Manifest header (named "Bundle-DocURL") identifying the bundle's documentation URL, from which further information about the bundle may be obtained.

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

4.8.12.8 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

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

Default value for the Bundle-Localization manifest header.

See Also BUNDLE_LOCALIZATION[p.132]

Since 1.3

4.8.12.10 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

4.8.12.11 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.

4.8.12.12 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.

4.8.12.13 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 ...
```

4.8.12.14 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 ...
```

4.8.12.15 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” ...
```

4.8.12.16 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 ...
```

4.8.12.17 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

4.8.12.18 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

4.8.12.19 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

4.8.12.20 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.

4.8.12.21 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.

4.8.12.22 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.

4.8.12.23 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

4.8.12.24 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

4.8.12.25 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

4.8.12.26 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.

4.8.12.27 **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.

4.8.12.28 **public static final String FRAGMENT_ATTACHMENT_ALWAYS = "always"**

Manifest header directive 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 directive 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_DIRECTIVE`[p.136]

Since 1.3

4.8.12.29 **public static final String FRAGMENT_ATTACHMENT_DIRECTIVE = "fragment-attachment"**

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

The directive 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.136], `Constants.FRAGMENT_ATTACHMENT_RESOLVETIME`[p.136], `Constants.FRAGMENT_ATTACHMENT_NEVER`[p.136]

Since 1.3

4.8.12.30 **public static final String FRAGMENT_ATTACHMENT_NEVER = "never"**

Manifest header directive 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 directive 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_DIRECTIVE`[p.136]

Since 1.3

4.8.12.31 **public static final String FRAGMENT_ATTACHMENT_RESOLVETIME =**

“resolve-time”

Manifest header directive 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 directive 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_DIRECTIVE[p.136]

Since 1.3

4.8.12.32 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

4.8.12.33 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

4.8.12.34 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.

4.8.12.35 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.

4.8.12.36 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.

4.8.12.37 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.

4.8.12.38 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.

4.8.12.39 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.

4.8.12.40 public static final String GROUPING_DIRECTIVE = "grouping"

Deprecated use Constants.USES_DIRECTIVE[p.142]

4.8.12.41 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.

4.8.12.42 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.

4.8.12.43 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

4.8.12.44 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

4.8.12.45 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

4.8.12.46 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.

4.8.12.47 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"
```

4.8.12.48 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

4.8.12.49 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

4.8.12.50 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.140], Constants.RESOLUTION_OPTIONAL[p.140]

Since 1.3

4.8.12.51 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:="mandito-
ry"
```

See Also Constants.RESOLUTION_DIRECTIVE[p.140]

Since 1.3

4.8.12.52 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.140]

Since 1.3

4.8.12.53 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)";
...
```

Since 1.3

4.8.12.54 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.

4.8.12.55 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.

4.8.12.56 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.106]) and followed by a dot (.). A bundle may use this as the prefix of the persistent identifiers for the services it registers.

4.8.12.57 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.121] method: If more than one service implements the specified class, the ServiceReference object with the highest ranking is returned.

The default ranking is zero (0). A service with a ranking of Integer.MAX_VALUE is very likely to be returned as the default service, whereas a service with a ranking of Integer.MIN_VALUE is very unlikely to be returned.

If the supplied property value is not of type java.lang.Integer, it is deemed to have a ranking value of zero.

4.8.12.58 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.

4.8.12.59 public static final String SINGLETON_ATTRIBUTE = "singleton"

Deprecated use Constants.SINGLETON_DIRECTIVE[p.142]

4.8.12.60 public static final String SINGLETON_DIRECTIVE = "singleton"

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

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

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

Since 1.3

4.8.12.61 public static final String SYSTEM_BUNDLE_LOCATION = "System Bundle"

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

4.8.12.62 public static final String USES_DIRECTIVE = "uses"

Manifest header directive (named "uses") identifying a list of packages that an exported package uses.

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

```
Export-Package: org.osgi.util.tracker; uses:="org.os-
gi.framework"
```

Since 1.3

4.8.12.63 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

4.8.12.64 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.143], Constants.VISIBILITY_REEXPORT[p.143]

Since 1.3

4.8.12.65 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.142]

Since 1.3

4.8.12.66 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.142]

Since 1.3

4.8.13 public interface Filter

An RFC 1960-based Filter.

Filter objects can be created by calling BundleContext.createFilter[p.118] with the chosen filter string.

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

Some examples of LDAP filters are:

```
    "(cn=Babs Jensen)"
    "(!(cn=Tim Howes))"
    "(&(" + Constants.OBJECTCLASS +
"=Person)(|(sn=Jensen)(cn=Babs J*)))"
    "(o=univ*of*mich*)"
```

Since 1.1

4.8.13.1 public boolean equals(Object obj)

obj The object to compare against this Filter object.

□ Compares this Filter object to another object.

Returns If the other object is a Filter object, then returns this.toString().equals(obj.toString();false otherwise.

4.8.13.2 public int hashCode( )

- ☐ Returns the hashCode for this Filter object.

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

4.8.13.3 public boolean match(ServiceReference reference)

reference The reference to the service whose properties are used in the match.

- ☐ Filter using a service's properties.

The filter is executed using properties of the referenced service.

Returns true if the service's properties match this filter; false otherwise.

4.8.13.4 public boolean match(Dictionary dictionary)

dictionary The Dictionary object whose keys are used in the match.

- ☐ Filter using a Dictionary object. The Filter is executed using the Dictionary object's keys and values.

Returns true if the Dictionary object's keys and values match this filter; false otherwise.

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

4.8.13.5 public boolean matchCase(Dictionary dictionary)

dictionary The Dictionary object whose keys are used in the match.

- ☐ Filter with case sensitivity using a Dictionary object. The Filter is executed using the Dictionary object's keys and values. The keys are case sensitivley matched with the filter.

Returns true if the Dictionary object's keys and values match this filter; false otherwise.

Since 1.3

4.8.13.6 public String toString( )

- ☐ Returns this Filter object's filter string.

The filter string is normalized by removing whitespace which does not affect the meaning of the filter.

Returns Filter string.

4.8.14 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.

4.8.14.1 **public static final int ERROR = 2**

An error has occurred.

There was an error associated with a bundle.

The value of ERROR is 0x00000002.

4.8.14.2 **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

4.8.14.3 **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 `PackageAdmin.refreshPackages`

Since 1.2

4.8.14.4 **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 `StartLevel`

4.8.14.5 **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 `StartLevel`

Since 1.2

4.8.14.6 **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

4.8.14.7 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.

4.8.14.8 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.

4.8.14.9 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.

4.8.14.10 public Throwable getThrowable( )

☐ Returns the exception related to this event.

Returns The related exception or null if none.

4.8.14.11 public int getType( )

☐ Returns the type of framework event.

The type values are:

- `STARTED`[p.145]
- `ERROR`[p.144]
- `WARNING`[p.145]
- `INFO`[p.145]
- `PACKAGES_REFRESHED`[p.145]
- `STARTLEVEL_CHANGED`[p.145]

Returns The type of state change.

4.8.15 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.117] method. FrameworkListener objects are called with a FrameworkEvent objects when the Framework starts and when asynchronous errors occur.

See Also `FrameworkEvent`[p.144]

4.8.15.1 public void frameworkEvent(FrameworkEvent event)

event The FrameworkEvent object.

□ Receives notification of a general FrameworkEvent object.

4.8.16 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.143] for a description of the filter string syntax.

4.8.16.1 public InvalidSyntaxException(String msg, String filter)

msg The message.

filter The invalid filter string.

□ Creates an exception of type InvalidSyntaxException.

This method creates an InvalidSyntaxException object with the specified message and the filter string which generated the exception.

4.8.16.2 public InvalidSyntaxException(String msg, String filter, Throwable cause)

msg The message.

filter The invalid filter string.

cause The cause of this exception.

□ Creates an exception of type InvalidSyntaxException.

This method creates an InvalidSyntaxException object with the specified message and the filter string which generated the exception.

Since 1.3

4.8.16.3 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

4.8.16.4 public String getFilter( )

□ Returns the filter string that generated the InvalidSyntaxException object.

Returns The invalid filter string.

See Also `BundleContext.getServiceReferences`[p.122], `BundleContext.addServiceListener(ServiceListener,String)`[p.117]

4.8.16.5 public Throwable initCause(Throwable cause)

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

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

Since 1.3

4.8.17 public final class PackagePermission extends BasicPermission

A bundle's authority to import or export a package.

A package is a dot-separated string that defines a fully qualified Java package.

For example:

org.osgi.service.http

PackagePermission has two actions: EXPORT and IMPORT. The EXPORT action implies the IMPORT action.

4.8.17.1 public static final String EXPORT = "export"

The action string export.

4.8.17.2 public static final String IMPORT = "import"

The action string import.

4.8.17.3 public PackagePermission(String name, String actions)

name Package name.

actions EXPORT,IMPORT (canonical order).

□ Defines the authority to import and/or export a package within the OSGi environment.

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

```
org.osgi.service.http
javax.servlet.*
*
```

Package Permissions are granted over all possible versions of a package. A bundle that needs to export a package must have the appropriate PackagePermission for that package; similarly, a bundle that needs to import a package must have the appropriate PackagePermssion for that package.

Permission is granted for both classes and resources.

4.8.17.4 public boolean equals(Object obj)

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

□ Determines the equality of two PackagePermission objects. This method checks that specified package has the same package name and PackagePermission actions as this PackagePermission object.

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

4.8.17.5 public String getActions( )

- Returns the canonical string representation of the PackagePermission actions.

  Always returns present PackagePermission actions in the following order: EXPORT,IMPORT.

Returns Canonical string representation of the PackagePermission actions.

4.8.17.6 public int hashCode( )

- Returns the hash code value for this object.

Returns A hash code value for this object.

4.8.17.7 public boolean implies(Permission p)

p The target permission to interrogate.

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

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

  The permission to export a package implies the permission to import the named package.

```
x.y.*,"export" -> x.y.z,"export" is true
*,"import" -> x.y, "import"      is true
*,"export" -> x.y, "import"      is true
x.y,"export" -> x.y.z, "export"  is false
```

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

4.8.17.8 public PermissionCollection newPermissionCollection( )

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

Returns A new PermissionCollection object.

4.8.18 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.151]

4.8.18.1 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.155]

4.8.18.2 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.124]

4.8.18.3 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.156], `BundleContext.ungetService`[p.125]

4.8.18.4 public ServiceEvent(int type, ServiceReference reference)

type The event type.

reference A ServiceReference object to the service that had a lifecycle change.

□ Creates a new service event object.

4.8.18.5 public ServiceReference getServiceReference( )

□ Returns a reference to the service that had a change occur in its lifecycle.

This reference is the source of the event.

Returns Reference to the service that had a lifecycle change.

4.8.18.6 public int getType( )

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

- `REGISTERED`[p.150]
- `MODIFIED`[p.149]
- `UNREGISTERING`[p.150]

Returns Type of service lifecycle change.

4.8.19 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.120]

4.8.19.1 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.120]

4.8.19.2 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.125]

4.8.20 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 Bundle-Context.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.149], ServicePermission[p.152]

4.8.20.1 public void serviceChanged(ServiceEvent event)

event The ServiceEvent object.

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

4.8.21 public final class ServicePermission extends BasicPermission

Indicates a bundle's authority to register or get a service.

- The ServicePermission.REGISTER action allows a bundle to register a service on the specified names.
- The ServicePermission.GET action allows a bundle to detect a service and get it.

ServicePermission to get the specific service.

4.8.21.1 public static final String GET = "get"

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

4.8.21.2 public static final String REGISTER = "register"

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

4.8.21.3 public ServicePermission(String name, String actions)

name class name

actions get,register (canonical order)

□ Create a new ServicePermission.

The name of the service is specified as a fully qualified class name.

```
ClassName ::= <class name> | <class name ending in ".*">
```

Examples:

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

There are two possible actions: get and register. The get permission allows the owner of this permission to obtain a service with this name. The register permission allows the bundle to register a service under that name.

4.8.21.4 public boolean equals(Object obj)

obj The object to test for equality.

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

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

4.8.21.5 public String getActions( )

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

Returns The canonical string representation of the actions.

4.8.21.6 public int hashCode( )

□ Returns the hash code value for this object.

Returns Hash code value for this object.

4.8.21.7 public boolean implies(Permission p)

p The target permission to check.

□ Determines if a ServicePermission object "implies" the specified permission.

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

4.8.21.8 public PermissionCollection newPermissionCollection( )

□ Returns a new PermissionCollection object for storing ServicePermission objects.

Returns A new PermissionCollection object suitable for storing ServicePermission objects.

4.8.22 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.121], `BundleContext.getServiceReferences`[p.122], `BundleContext.getService`[p.120]

4.8.22.1 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.124]

4.8.22.2 public Object getProperty(String key)

key The property key.

□ Returns the property value to which the specified property key is mapped in the properties Dictionary object of the service referenced by this ServiceReference object.

Property keys are case-insensitive.

This method must continue to return property values after the service has been unregistered. This is so references to unregistered services (for example, ServiceReference objects stored in the log) can still be interrogated.

Returns The property value to which the key is mapped; null if there is no property named after the key.

4.8.22.3 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.124] or ServiceRegistration.setProperties[p.155] methods.

Returns An array of property keys.

4.8.22.4 public Bundle[] getUsingBundles( )

□ Returns the bundles that are using the service referenced by this ServiceReference object. Specifically, this method returns the bundles whose usage count for that service is greater than zero.

Returns An array of bundles whose usage count for the service referenced by this ServiceReference object is greater than zero; null if no bundles are currently using that service.

Since 1.1

4.8.22.5 public boolean isAssignableTo(Bundle bundle, String className)

bundle the `Bundle` object to check the package source for

className the class name to check the package source for

□ Returns true if the bundle which registered the service referenced by this `ServiceReference` and the specified bundle use the same source for the package of the specified class name.

This method performs the following checks:
1) Get the package name from the className
2) For the bundle that registered the service referenced by this `ServiceReference` object (registrant bundle); find the source for the package. If no source is found then return true if the registrant bundle is equal to the specified bundle; otherwise return false;
3) If the package source of the registrant bundle is equal to the package source of the specified bundle then return true; otherwise return false.

Returns true if the bundle which registered the service referenced by this `ServiceReference` and the specified bundle use the same source for the package of the specified class name; otherwise false is returned.

4.8.23 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.124]

4.8.23.1 public ServiceReference getReference( )

□ Returns a ServiceReference object for a service being registered.

The ServiceReference object may be shared with other bundles.

Returns ServiceReference object.

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

4.8.23.2 public void setProperties(Dictionary properties)

properties The properties for this service. See `Constants`[p.131] 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.139] and Constants.SERVICE_ID[p.141] 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.149] is synchronously sent.

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

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

4.8.23.3 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.150] 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.150] 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.125], ServiceFactory.ungetService[p.151]

4.8.24 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.117] method. SynchronousBundleListener objects are called with a BundleEvent object when a bundle has been installed, started, stopped, updated, or uninstalled.

Unlike normal BundleListener objects, SynchronousBundleListener s are synchronously called during bundle life cycle processing. The bundle life cycle processing will not proceed until all SynchronousBundleListener s 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.126]

Since 1.1

4.8.25 public class Version implements Comparable

Version identifier for bundles and packages.

Version identifiers have four components.

1 Major version. A non-negative integer.
2 Minor version. A non-negative integer.
3 Micro version. A non-negative integer.
4 Qualifier. A text string. See Version(String) for the format of the qualifier string.

Version instances are immutable.

Since 1.3

4.8.25.1 public static final Version emptyVersion

The empty version “0.0.0”. Equivalent to calling new Version(0,0,0).

4.8.25.2 public Version(int major, int minor, int micro)

major Major component of the version identifier.

minor Minor component of the version identifier.

micro Micro component of the version identifier.

□ Creates a version identifier from the specified numerical components.

The qualifier is set to the empty string.

Throws `IllegalArgumentException` – If the numerical components are negative.

4.8.25.3 public Version(int major, int minor, int micro, String qualifier)

major Major component of the version identifier.

minor Minor component of the version identifier.

micro Micro component of the version identifier.

qualifier Qualifier component of the version identifier. If null is specified, then the qualifier will be set to the empty string.

□ Creates a version identifier from the specifed components.

Throws `IllegalArgumentException` – If the numerical components are negative or the qualifier string is invalid.

4.8.25.4 public Version(String version)

version String representation of the version identifier.

□ Created a version identifier from the specified string.

Here is the grammar for version strings.

```
version ::= major('.'minor('.'micro('.'qualifier)?)?)?
major ::= digit+
minor ::= digit+
micro ::= digit+
qualifier ::= (alpha|digit|'_'|'-')+
digit ::= [0..9]
alpha ::= [a..zA..Z]
```

Throws `IllegalArgumentException` – If version is improperly formatted.

4.8.25.5 public int compareTo(Object object)

object The Version object to be compared.

□ Compares this Version object to another object.

A version is considered to be **less than** another version if its major component is less than the other version's major component, or the major components are equal and its minor component is less than the other version's minor component, or the major and minor components are equal and its micro component is less than the other version's micro component, or the major, minor and micro components are equal and it's qualifier component is less than the other version's qualifier component (using String.compareTo).

A version is considered to be **equal to** another version if the major, minor and micro components are equal and the qualifier component is equal (using String.compareTo).

Returns A negative integer, zero, or a positive integer if this object is less than, equal to, or greater than the specified Version object.

Throws `ClassCastException` – If the specified object is not a Version.

4.8.25.6 public boolean equals(Object object)

object The Version object to be compared.

□ Compares this Version object to another object.

A version is considered to be **equal to** another version if the major, minor and micro components are equal and the qualifier component is equal (using String.equals).

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

4.8.25.7 public int getMajor( )

□ Returns the major component of this version identifier.

Returns The major component.

4.8.25.8 public int getMicro( )

□ Returns the micro component of this version identifier.

Returns The micro component.

4.8.25.9 public int getMinor( )

□ Returns the minor component of this version identifier.

Returns The minor component.

4.8.25.10 public String getQualifier( )

□ Returns the qualifier component of this version identifier.

Returns The qualifier component.

4.8.25.11 public int hashCode( )

□ Returns a hash code value for the object.

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

4.8.25.12 public static Version parseVersion(String version)

version String representation of the version identifier. Leading and trailing whitespace will be ignored.

□ Parses a version identifier from the specified string.

See Version(String) for the format of the version string.

Returns A Version object representing the version identifier. If version is null or the empty string then emptyVersion will be returned.

Throws `IllegalArgumentException` – If version is improperly formatted.

4.8.25.13 public String toString( )

□ Returns the string representation of this version identifier.

The format of the version string will be major.minor.micro if qualifier is the empty string or major.minor.micro.qualifier otherwise.

Returns The string representation of this version identifier.

4.9 References

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

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

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

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

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

[37] *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

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

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

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

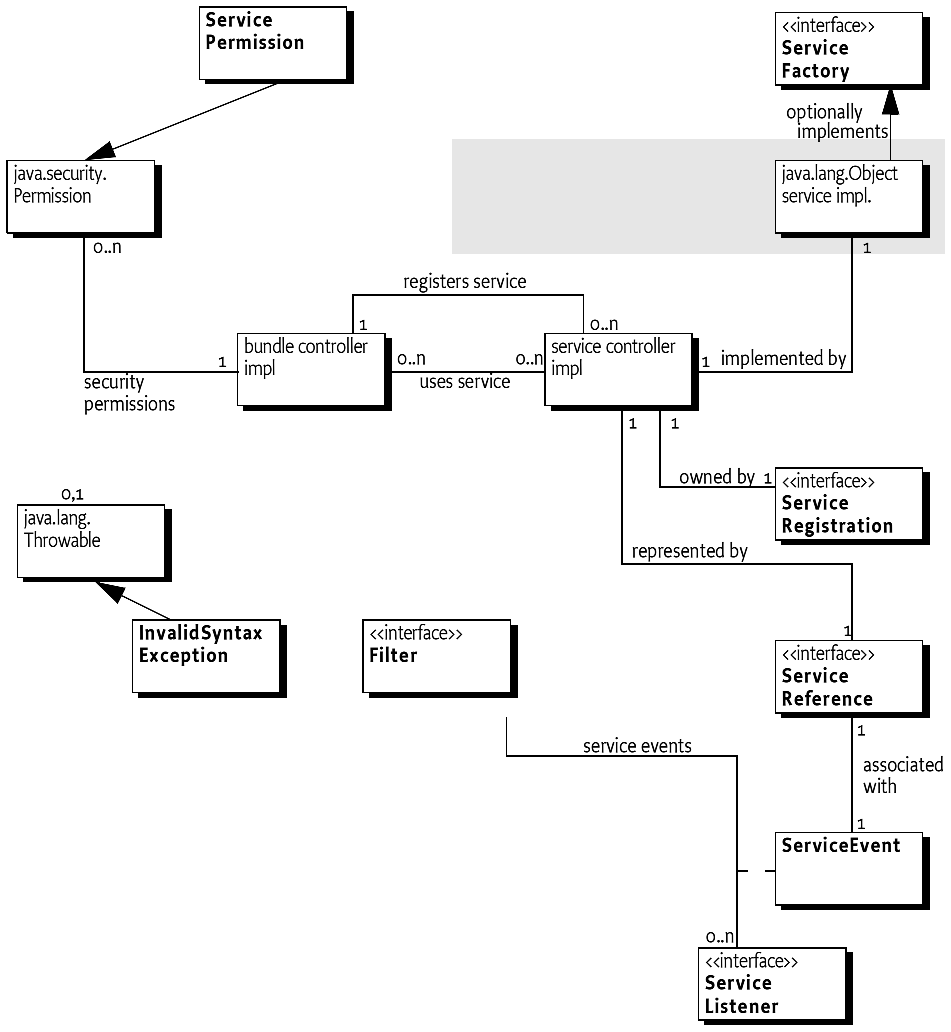
[41] *W3C EBNF*
http://www.w3c.org/TR/REC-xml#sec-notation

[42] *Lexical Structure Java Laguage*
http://java.sun.com/docs/books/jls/second_edition/html/lexical.doc.html

[43] *Interval Notation*
http://www.math.ohio-state.edu/courses/math104/interval.pdf

5 Service Layer

Figure 37 *Class Diagram* org.osgi.framework



5.1 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.

5.1.1 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 167.

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.

5.1.2 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.

5.1.3 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 170.

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 *Resolving* on page 37. 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.

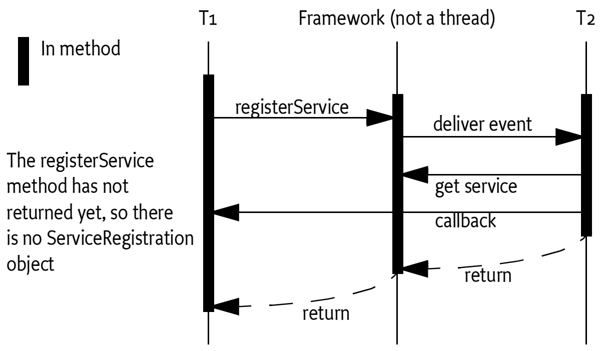
If the service object is successfully registered, the Framework must return a ServiceRegistration object to the caller. A service object can be unregistered only by the holder of its ServiceRegistration object (see the unregister() method). Every successful service object registration must yield a unique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

5.1.4 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 38 on page 164 shows such a sequence.

Figure 38 *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.

5.1.5 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 169 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 10 Standard Framework Service Registry Properties contains a list of pre-defined properties.

Property Key	Type	Constants	Property Description
objectClass	String[ ]	OBJECTCLASS	The objectClass property contains the set of class and interface names under which a service object is registered with the Framework. The Framework must set this property automatically. The Framework must guarantee that when a service object is retrieved with BundleContext.getService(ServiceReference), it can be cast to any of these classes or interfaces.
service.description	String	SERVICE_DESCRIPTION	The service.description property is intended to be used as documentation and is optional. Frameworks and bundles can use this property to provide a short description of a registered service object. The purpose is mainly for debugging because there is no support for localization.

Table 10 Standard Framework Service Registry Properties

Property Key	Type	Constants	Property Description
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.
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 10 Standard Framework Service Registry Properties

5.1.6 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 13 for more information.

5.1.7 Obtaining Services

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

- getServiceReference(String) – This method returns a ServiceReference object to a service object that implements, and was registered under, the name of the service interface specified as String. If multiple such service objects exist, the service object with the highest SERVICE_RANKING is returned. If there is a tie in ranking, the service object with the lowest SERVICE_ID (the service object that was registered first) is returned.
- getServiceReferences(String,String) – This method returns an array of ServiceReference objects that:
  - Implement and were registered under the given service interface.
  - Satisfy the search filter specified. The filter syntax is further explained in *Filters* on page 169.

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.

5.1.8 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.

5.1.9 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 170 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.

5.1.10 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.

5.2 Service Events

- ServiceEvent – Reports registration, unregistration, and property changes for service objects. All events of this kind must be delivered synchronously. The type of the event is given by the getType method, which returns an int. Event types can be extended in the future, unknown events should be ignored.
- 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.

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 263).

5.3 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 263.

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 170.
- 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.

5.4 Filters

### Case sensitive check method.

The Framework provides a Filter interface, and uses a search filter syntax in the getServiceReference(s) method that is defined in *Filter Syntax* on page 33. Filter objects can be created by calling createFilter(String) with the chosen filter string. The filter supports to match methods:

- match(ServiceReference) – Match the properties of the Service Reference in a case insensitive way.
- match(Dictionary) – Match the entries in the given Dictionary object in a case insensitive way. The comparison must follow the rules as defined in the String.compareToIgnoreCase.
- matchCase(Dictionary) – Match the entries in the given Dictionary object in a case-sensitive way.

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

This matching requires comparing the value string in the filter to a target object from the service property or dictionary. This comparison can be executed with the Comparable interface if the target object's class implements a constructor taking a single String object and the class implements the Comparable interface. I.e. if the target object is of class Target, the class Target must implement:

- A constructor Target(String)
- Implement the java.lang.Comparable interface

If the target object does not implement java.lang.Comparable, the =, ~=, <= >= operators must return only true when the objects are equal (using the equals(Object) method).

The following example shows how a class can veryify the ordering of an enumeration with a filter.

```
public class B implements Comparable {
   String keys[] = {"bugs", "daffy", "elmer", "pepe"};
   int     index;

   public B(String s) {
      for ( index=0; index<keys.length; index++ )
         if ( keys[index].equals(s) )
            return;
   }

   public int compareTo( Object other ) {
      V vother = (V) other;
      return index - vother.index;
   }
 }
```

The class could be used with the following filter:

```
(!(enum>=elmer))    -> matches bugs and daffy
```

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

5.5 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 163.
- 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.

5.6 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>
```

5.7 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.

5.8 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.

5.9 Security and Services

5.9.1 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 163.

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 162.

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.

5.10 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 23.

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.

5.11 Changes

This section defines the changes since Framework version 1.2.

5.11.1 Filter processing for unknown types

The framework filter processing has been extended to handle objects of unknown type if that type has a constructor which takes a single String argument.

5.11.2 Minor clarifications

•

5.12 References

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

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

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

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

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

[49] *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

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

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

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

[53] *W3C EBNF*
http://www.w3c.org/TR/REC-xml#sec-notation

[54] *Lexical Structure Java Laguage*
http://java.sun.com/docs/books/jls/second_edition/html/lexical.doc.html

[55] *Interval Notation*
http://www.math.ohio-state.edu/courses/math104/interval.pdf

6 Package Admin Service Specification

Version 1.1

6.1 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.

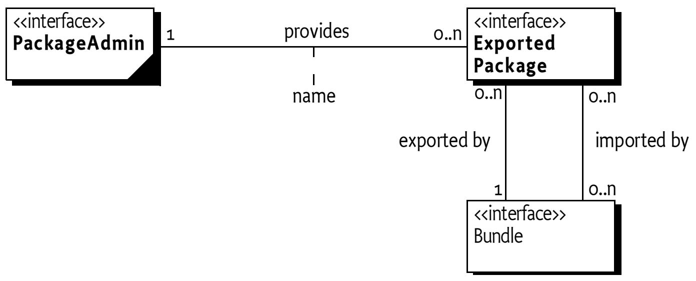
6.1.1 Essentials

- *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.

6.1.2 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 39 *Class Diagram org.osgi.service.packageadmin*



6.1.3 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 *Resolving* on page 37. 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.

6.2 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.

6.2.1 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 191.

6.3 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.

6.4 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.

6.5 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; version=1.2
```

6.5.1 Summary

- *ExportedPackage* - An exported package. [p.179]
- *PackageAdmin* - Framework service which allows bundle programmers to inspect the packages exported in the Framework and eagerly update or uninstall bundles. [p.180]
- *RequiredBundle* - A required bundle. [p.184]

6.5.2 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.

6.5.2.1 public Bundle getExportingBundle( )

- Returns the bundle exporting the package associated with this Exported-Package object.

Returns The exporting bundle, or null if this ExportedPackage object has become stale.

6.5.2.2 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 Exported-Package object are considered to be importing bundles and are included in the returned array. See RequiredBundle.getRequiringBundles()[p.185]

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

6.5.2.3 public String getName( )

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

Returns The name of this ExportedPackage object.

6.5.2.4 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.

6.5.2.5 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.

6.5.3 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.179])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.183] is called. If an Exported-Package 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.

6.5.3.1 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

6.5.3.2 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

6.5.3.3 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

6.5.3.4 public int getBundleType(Bundle bundle)

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

- BUNDLE_TYPE_FRAGMENT[p.181]

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

6.5.3.5 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.

6.5.3.6 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.

6.5.3.7 public ExportedPackage[] getExportedPackages(String name)

name The name of the exported packages to be returned.

□ Get 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

6.5.3.8 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. ### what do you get when the bundle is not resolved?

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

Since 1.2

6.5.3.9 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

6.5.3.10 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

6.5.3.11 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.

6.5.3.12 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

6.5.4 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

6.5.4.1 public Bundle getBundle( )

- Returns the bundle which defines this RequiredBundle.

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

6.5.4.2 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.

6.5.4.3 public String getSymbolicName( )

- Returns the symbolic name of the bundle.

Returns The symbolic name of the bundle.

6.5.4.4 public Version getVersion( )

- Returns the version of the bundle.

Returns The version of the bundle.

6.5.4.5 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.

7 Start Level Service Specification

Version 1.0

7.1 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.

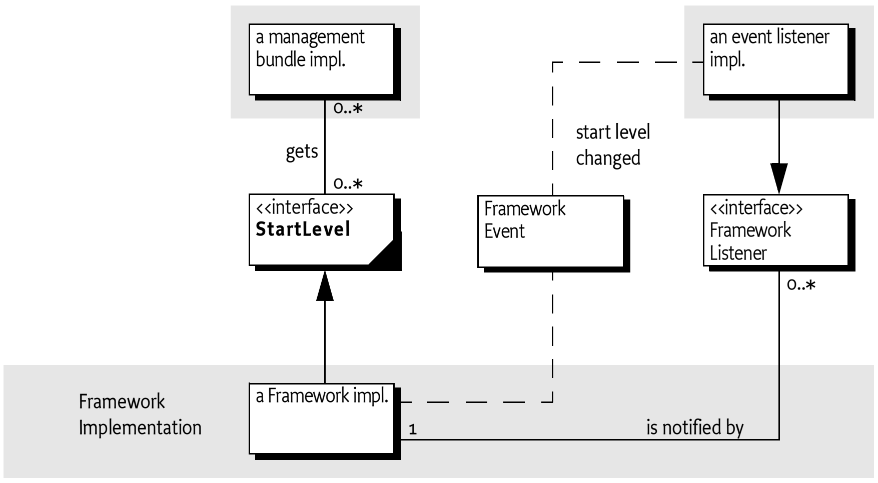
7.1.1 Essentials

- *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.

7.1.2 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 94.
- *Framework Listener* – See page 94.

Figure 40 Class Diagram org.osgi.service.startlevel package



7.2 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.

7.2.1 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.

7.2.2 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.

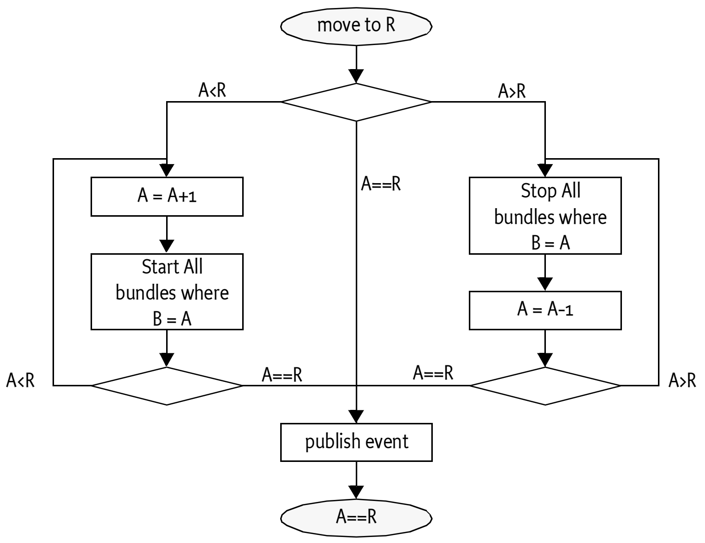
When the requested start level is reached and all bundles satisfy the condition that their bundle start level <= active start level in order to be started, then the FrameworkEvent.STARTLEVEL_CHANGED event must be sent to all registered FrameworkListener objects. If the requested start level and active 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 41.

Figure 41 *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.

7.2.3 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 189 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.

7.2.4 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 189 to make the active start level equal to zero.

7.2.5 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.

7.2.6 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.

7.2.7 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.

7.2.8 System Bundle

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

7.3 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.

7.4 Example Applications

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

7.4.1 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.

7.4.2 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() { ... }
}
```

7.5 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].

7.6 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; version=1.0
```

7.6.1 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.

7.6.1.1 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.

7.6.1.2 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.195]

7.6.1.3 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.

7.6.1.4 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.

7.6.1.5 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.

7.6.1.6 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.

7.6.1.7 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.

8 Conditional Permission Admin Specification

Version 1.0

8.1 Introduction

The OSGi security model is based on the powerful and flexible Java 2 security architecture, specifically the permission model. This specification adds the several new features to the Java 2 model to adapt it to the typical use cases of OSGi deployments.

In contrast with other execution models, the OSGi provides a well defined API to manage permissions, other models tend to leave the permission management up to implementations. A key aspect of this security API is the real time management of the permissions. This enables management applications to control the permissions of other applications with immediate effect; no restart is required.

Permission Management is based on the very general mode of *conditional permissions*. Conditional permissions match permissions to bundles using OSGi or user defined conditions. The advantage of this model is that groups of permissions can be shared based on signers, locations, and bundles that have a particular name. Conditions can also be used to enable a group of permissions when an external condition is true, e.g. an inserted SIM card, an online connection to the management system is established, or a user has approved a permission after prompting.This model allows an operator to create and enforce a dynamic security policy for its devices.

This specification defines a Conditional Permission Admin that supersedes the Permission Admin (albeit its relation to Permission Admin is well defined in this specification). It provides a security model based on conditional permissions where the conditions define the selection of the bundles that these permissions apply to.

8.1.1 Essentials

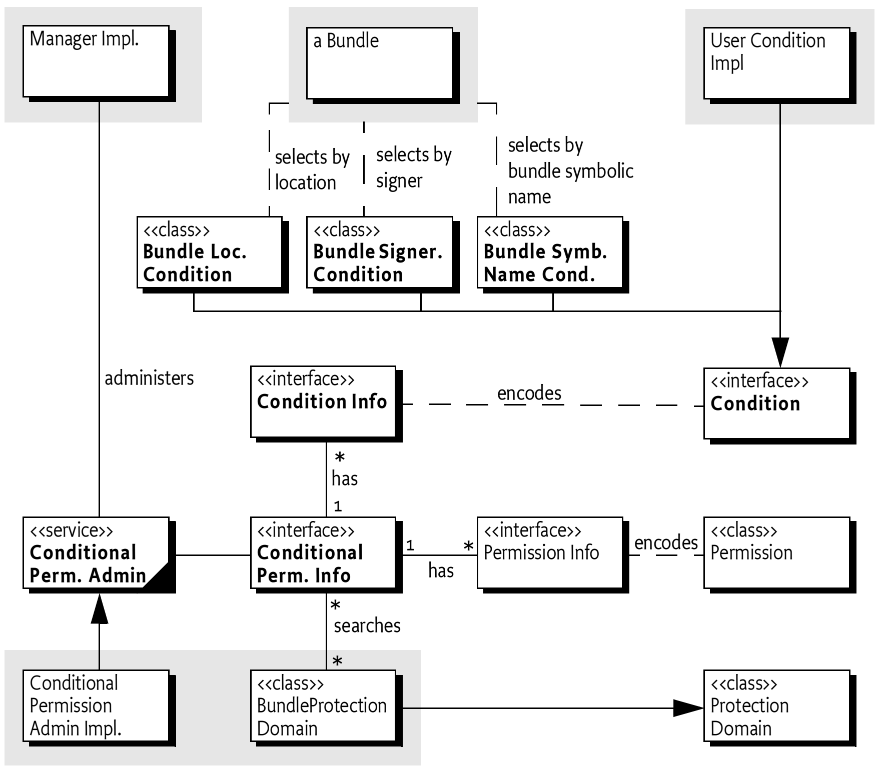
- *Policies* – Provide a security policy system where external conditions control the actual permissions that bundles have at a certain moment in time.
- *Java 2 Security* – Provide full compatibility with the existing Java 2 security model, existing must not require modifications.
- *Delegation* – Support a management delegation model where an Operator can delegate securely part of the management of a device to another party.
- *Digital Signatures* – Support the use of digital signatures in a bundle's policy decisions.

- *Real Time* – Changes in the environment must be reflected immediately in the bundle's permissions.
- *Operator Specific Conditions* – It must be possible for operators, manufacturers, selected developers, and others to provide custom conditions.
- *User Confirmation* – The policy model must support end user prompting and confirmations.
- *Backward Compatibility* – The model must be backward compatible with the Permission Admin of earlier releases.

8.1.2 Entities

- *Conditional Permission Admin* – The administrative service that provides the functions to manipulate the *permission table*.
- *Permission Table* – A conceptual table containining all the Conditional Permission Info tuples.
- *Conditional Permission Info* – A tuple of a set of ConditionInfo objects and a set of PermissionInfo objects.
- *Permission Info* – Holds a string based encoding of a Permission object.
- *Condition Info* – Holds a string based encoding of a Condition object.
- *Condition* – Condition objects is associated witha Bundle Protection Domain and abstract an external condition that can be evaluated at any time.
- *Bundle Location Condition* – An immutable Condition object that is satisfied when the associated bundle has the given location.
- *Bundle Signer Condition* – An immutable Condition object that is satisfied when the associated bundle is signed by a certificate that matched the given DN.
- *Bundle Symbolic Name Condition* – An immutable Condition object that is satisfied when the associated bundle's symbolic name matches the given symbolic name.
- *Permission* – An object that defines a certain permission type.
- Bundle Protection Domain – The class that implements the Protection Domain of a bundle, this specification does not define an interface for this class, but it plays an important role in this specification.

Figure 42 org.osgi.service.condpermadmin package


Manager Impl.
a Bundle
User Condition Impl
selects by location
selects by signer
selects by bundle symbolic name
<<class>> Bundle Loc. Condition
<<class>> Bundle Signer. Condition
<<class>> Bundle Symb. Name Cond.
administers
<<interface>> Condition Info
encodes
<<interface>> Condition
*
has
1
<<service>> Conditional Perm. Admin
<<interface>> Conditional Perm. Info
1
*
has
<<interface>> Permission Info
encodes
<<class>> Permission
*
searches
*
Conditional Permission Admin Impl.
<<class>> BundleProtection Domain
<<class>> Protection Domain

8.1.3 Synopsis

A Conditional Permission Admin service maintains a system wide table of ConditionalPermissionInfo objects, which are an encoded form of conditions and permissions. A manager can enumerate, delete, and add new tuples to this table.

When a bundle is created, it creates a single Bundle Protection Domain that instantiates the conditions and permissions defined in the permission table, potentially pruning any entries that can never apply to that bundle and optimizing entries that always apply.

A bundle can have local permissions carried in a Bundle Permission Resource. These are the fine grained permissions that this bundle needs to operate.

During the permission check of the Java Security Manager, it first checks the local permissions, if this fails, the check fails. Otherwise, the Bundle Protection Domains of the calling bundles are consulted to see if they imply the requested permission. To imply the requested permission, the Bundle Protection Domain must find a tuple in its permission table where all conditions are satisfied and where the tuple's permissions imply the requested permission. However, certain conditions must postpone their evaluation so they can be grouped. Such conditions are only postponed when their related permissions imply the requested permission and the request can not be satisfied immediately.

At the end of the permission check, the postponed conditions are evaluated grouped by their class.

8.1.4 What To Read

8.1.4.1 Architects

- *Permission Management Model* on page 202
- *Conditional Permissions* on page 209
- *Conditions* on page 221
- *Digitally Signed JAR Files* on page 15

8.1.4.2 Application Programmers

- *Standard Conditions* on page 228
- *Bundle Permission Resource* on page 230
- *Digitally Signed JAR Files* on page 15

8.1.4.3 Management Programmers

- *Permission Management Model* on page 202
- *Conditional Permissions* on page 209
- *Conditions* on page 221
- *Conditional Permissions* on page 209
- *Permission Management* on page 219
- *Bundle Permission Resource* on page 230
- *Digitally Signed JAR Files* on page 15

8.2 Permission Management Model

The Conditional Permission Admin provides an extremely flexible security model for bundles. However, the price of this flexibility is additional complexity. In this case, the amount of configuration necessary to setup a working system can easily overwhelm anybodt. It is therefore necessary to be very careful implementing a deployment security model. This section defines a series of possible deployment security models while simultaneously defining the terminology that is used in later sections that explain the available mechanisms in detail.

8.2.1 Local Permissions

A good working principle is to minimize permissions as much as possible, as early as possible. This principle is embodied with the *local permissions* of a bundle. Local permissions are defined by a Bundle Permission Resource that is embodied in the bundle; it defines a set of *permissions*. These permissions must be enforced by the Framework for the given bundle. I.e. a bundle can get less permissions than the local permissions but it can never get more permissions. If no such resource is present then the local permissions are assumed to be All Permission. The Bundle Permission Resource is defined in *Bundle Permission Resource* on page 230.

For example, if the local permissions do not imply ServicePermission[org.osgi.service.log.LogService,GET], then the bundle can never get the LogService object, regardless of any other security setup in the device.

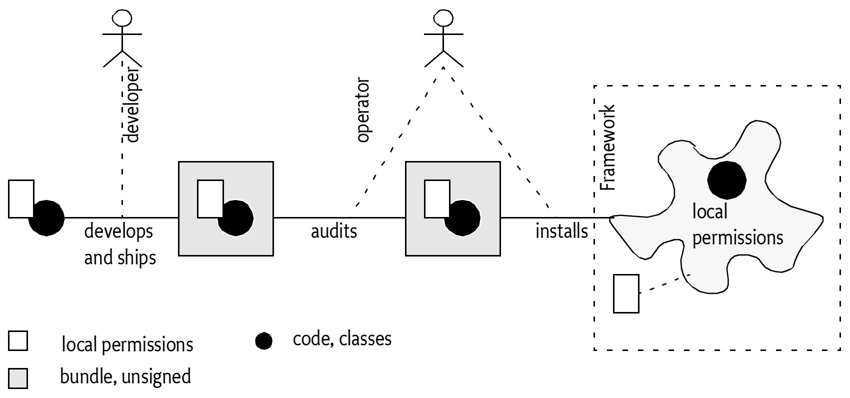
The fine grained permissions allowed by the OSGi Service Platform are very effective with the local permissions because they can be defined by the developer instead of the deployer. The developer knows exactly what services are needed, what packages the bundle requires and what network hosts are accessed. Tools can be used that analyze bundles and provide the appropriate local permissions to simplify the task of the developer. However, without detailed knowledge of the bundle's intrinsics, it is very difficult to create the local permissions due to their fine grained granularity.

At first sight, it can seem odd that a bundle carries its own permissions. However, the local permissions define the *maximum* permissions that the bundle needs, providing more permissions to the bundle is irrelevant because the Framework must not allow the bundle to use them. The purpose of the local permissions is therefore *auditing* by the deployer. Analyzing a bundle's byte codes for its security requirements is cumbersome, if not impossible. Auditing a bundle's permission resource is (relatively) straightforward. For example, if the local permissions request permission to access the Internet, it is clear that the bundle has the potential to access the network. By inspecting the local permissions, the Operator can quickly see the security scope of the bundle. It can trust this audit because it must be enforced by the Framework when the bundle is executed.

An Operator that runs a fully closed system can use the local permissions to run third party applications that are not trusted to run unchecked, thereby mitigating risks. The Framework guarantees that a bundle is never granted a permission that is not implied by its local permissions. A simple audit of the application's local permissions will reveal any potential threats.

This scenario is depicted in Figure 43. A developer creates a bundle with local permissions, the operator verifies the local permissions, and if it matches the expectations, it is deployed to the device where the Framework verifies that the local permissions are never exceeded.

Figure 43 *Local permissions and Deployment*



Summarizing, the benefits of local permissions are:

- *Fine grained* – The developer has the knowledge to provide the fine grained permissions that are necessary to minimize the sandbox of the bundle without constraining it.
- *Auditable* – The Operator has a relatively small and readable file that shows the required sandbox. It can therefore asses the risk of running a bundle.
- *Sandboxed* – The Operator has the guarantee from the Framework that a bundle can not escape its local permissions.

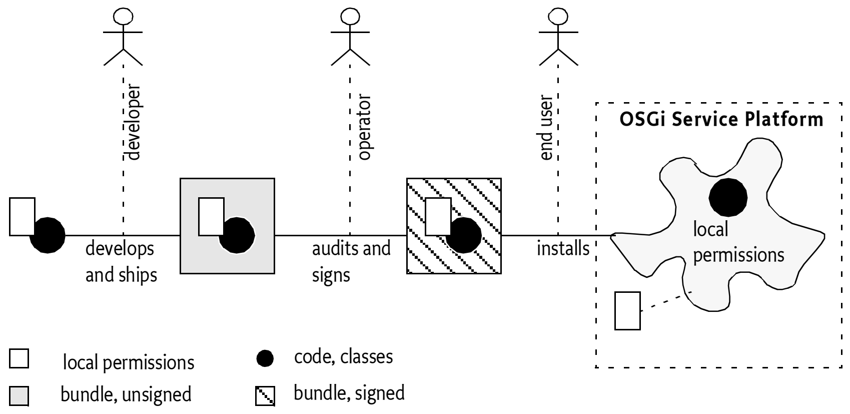
8.2.2 Open Deployment Channels

From a business perspective it is sometimes too restrictive to maintain a fully closed system.There are many use cases where users should be able to deploy bundles from a CD, via a PC, or from an Internet web sites. In those scenarios relying on the local permissions is not sufficient because the Framework can not verify that the local permissions have not been tampered with.

The de facto solution to tampering is to *digitally sign* the bundles. The rules for OSGi signing are defined in *Digitally Signed JAR Files* on page 15. A digital signing algorithms detects modifications of the JAR as well as authenticating the signer. A Framework therefore must refuse to run a bundle when a signature does not match the contents or it does not recognize the signer. Signing therefore makes it possible to use an untrusted deployment channel and still rely on the enforcement of the local permissions.

For example, an Operator can provision its applications via the Internet. When such an application is downloaded from an untrusted site, the Framework verifies the signature. Only when the signature is trusted, it installs the application and uses the local permissions of the bundle to sandbox it.

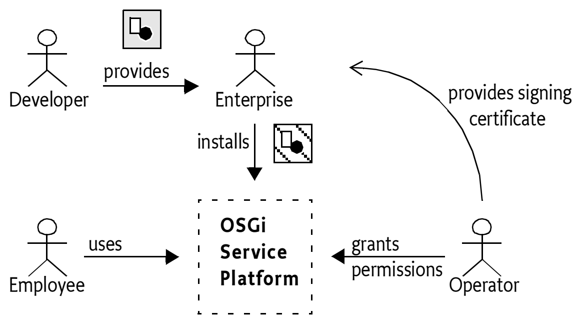
Figure 44 *Local Scope and Deployment with signing*



8.2.3 Delegation

A model where the local permissions are secured with only a signature works for an Operator that fully controls the device. I.e. the operator must still verify the local permissions and sign every bundle before it is provisioned. This can become expensive when there are third parties involved. For example, an Enterprise could provide applications to its employees on a mobile phone that is managed by an Operator. This model is depicted in Figure 45. If the Enterprise always has to contact the Operator before it can provision a new version, bottlenecks quickly arise.

Figure 45 *Delegation model*



This bottleneck problem can also be solved with signing. Signing does not only provide tamper detection, it also provides an authenticated *principal*. The principal is authenticated with a certificate chain. The device contains a set of trusted certificates (depending on implementation) that are used to authenticate the certificate of the signer.

The operator can therefore safely associate a principal with a set of permissions. These permissions are called the *system permissions*. Bundles signed by that principal are then automatically granted those system permissions.

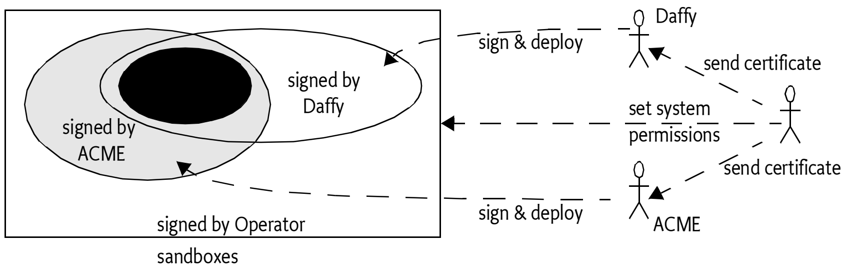
In this model, the Operator is still fully in control. At any moment in time, the Operator can change the system permissions associated with the principal and thereby immediately deny access to all bundles of that principal, while they are running. Alternatively, the Operator can add additional system permissions to the principal if a new service has become available to the signer's applications. For example, if the Operator installs a `org.tourist.PointOfInterest` service, it can grant the `ServicePermission[org.tourist.PointOfInterest,GET]` and `PackagePermission[org.tourist,IMPORT]` to all principals that are allowed to use this service. The Operator can inform the involved parties after the fact, if at all. This model therefore does not create a bottleneck.

Using digital signing to assign system permissions can therefore *delegate* the responsibility of provisioning to other parties. The Operator completely defines the limits of the permissions of the principal, but the signing and deployment can be done by the other parties.

For example, an Operator can define that the ACME company can provision bundles without any intervention of the Operator. The Operator has to provide ACME once with a signing certificate and the Operator must associate the ACME principal with the appropriate system permissions on the device.

The key advantage of this model is the reduced communication between ACME and the Operator: The Operator can modify the system permissions of ACME applications and be in control at any moment in time. The ACME company can develop new applications without the need to coordinate these efforts in any way.This model is depicted in Figure 46.

Figure 46 *Typical Delegation model*



The local permissions can still play an important role in the delegation model because it provides the signer the possibility to mitigate its risk, just like it did for the Operator. Signers can verify the local permissions before they sign a bundle. Like the Operator in the earlier scenario, the signer can quickly verify the security requirements of a bundle. For example, if a game bundle requests `AdminPermission[*,*]`, it is likely that the bundle will not pass the security audit of the signer. However, in the unlikely case it did, it will not be granted this permissions unless the Operator gave such a permission to the signer's principal on the device.

8.2.4 Grouping

### not finished, not sure it is needed. It is a powerful model, and well known. However, the local permissions provide a more powerful model and overlap with using signing + grouping.

The grouping model is traditionally used because it minimizes the administration of the security setup. For example, an operator can define the following security levels:

- *Untrusted* – Applications that are not really trusted. These applications must run in a very limited security scope. They could be unsigned.
- *Trusted* – Applications that are basically trusted but that are not allowed to manage the device or provide system services.
- *System* – Applications that provide system services.
- *Manage* – Applications that manage the device.

The operator signs the bundle with an appropriate certificate before it is deployed, when the device is installed, it will be automatically assigned to the appropriate security scope.

However, the local scope of the similar behavior can be obtained with the local scope of a bundle.

8.2.5 Typical Example

This example provides a simple setup for a delegation model. The example is intended for readability, certain concepts will be explained later. Also for readability, package prefixes that can be easily guessed are replaced with ....

Basic permissions define the permissions that are available to all bundles. The basic permissions therefore have no conditions associated with them so all bundles will be able to use these permissions:

```
{
   (..ServicePermission ..LogService GET )
   (..PackagePermission ..log IMPORT )
   (..PackagePermission ..framework IMPORT )
}
```

The next permission tuple has a condition that limits the permissions to bundles that are signed by ACME. ACME signed bundles are given the permission to manage other ACME bundles and to configure other ACME bundles as well as becoming configurable. Conditions are encapsulated in square brackets ('[]').

```
{
   [ ..BundleSignerCondition "* ; o=ACME, c=GB" ]
   ( ..AdminPermission "(signer=ou=Signer, o=ACME)" "*" )
```

### Needs update if we know how to handle this

```
   ( ..ServicePermission ..ManagedService* REGISTER )
   ( ..PackagePermission ..cm IMPORT )
   ( ..ConfigurationPermission "(signer=cn=ACME)"
        SET,GET )
```

### The signer with Configuration Permission is under discussion...

```
}
```

The last permission tuple is for bundles signed by the operator. The operator bundles get full managing capabilities as well as permissions to provide system services.

```
{
   [ ..BundleSigner "*, o=Operator, c=FR" ]
   ( ..AdminPermission "*" "*" )
   ( ..ServicePermission * "GET,REGISTER" )
   ( ..PackagePermission * "IMPORT,EXPORT" )
}
```

The resulting permissions are summarized in Table 11 on page 208.

8.3 Effective Permissions

Once a bundle is installed, it has Java 2 *permissions* associated with it. This set is called the *effective permissions*. The effective permissions are consulted when the checkPermission method of the Security Manager is called.

Signed by:			Unsigned	ACME	Operator
Service Permission	..LogService	GET	x	x	x/i
	..ManagedService∗	REGISTER		x	i
		GET			i
	∗	∗			x
PackagePermission	..log	IMPORT	x	x	x/i
	..cm	IMPORT		x	i
	..framework	IMPORT	x	x	x/i
	∗	∗			x
AdminPermission	(signer=cn=ACME)	∗		x	i
	∗	∗			x

Table 11 Assigned Permission, implied by another permission, x=set

The Permission Admin service and the Conditional Permission Admin service can be used by an application to define the *system permissions*. Additionally, a bundle can carry its own permissions, called the *local permissions*. All these permission sets interact in a non trivial way to give the effective permissions.

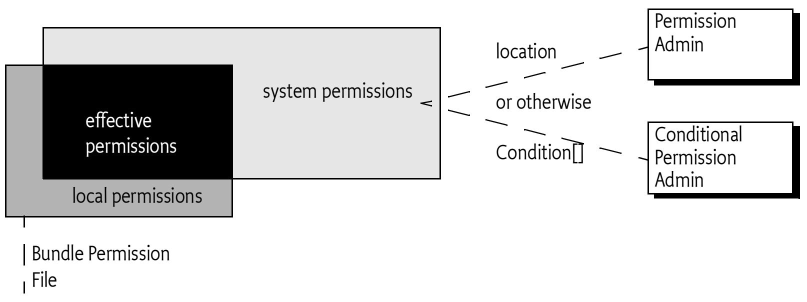
The purpose of the local permissions are to mitigate the risk for the signer of the bundle. The Framework guarantees that a bundle's effective permissions are always smaller or equal than the local permissions because the effective permissions are the intersection of the local permissions with the system permissions.

$$Effective = Local \cap System$$

The system permissions have two possible sources. The system permissions can be bound via Permission Admin to a location. This mechanism is provided for backward compatibility only. New management applications should use the Conditional Permission Admin if possible.

If the Permission Admin location is not bound, all the *conditional permissions* from Conditional Permission Admin act as the system permissions. The relationship between the system permissions and local permissions is depicted in Figure 47.

Figure 47 *System, Local and Security permissions*



### I assume that the intersection takes place also when the location is set through Permission Admin? Otherwise this guarantee cannot be given

### I assume the Bundle Permission Resource cannot have conditions?

### Why is Permission Admin not modelled with a BundleLocationCondition? I realize that this has a slighly different behavior but is that a problem in practice? Especially when a manager ONLY uses the Permission Admin?

8.4 Conditional Permissions

The conditional permissions provide a very general model that is related to, but quite different from the Java 2 Policy model. Permissions are no longer grouped per bundle (or by protection domain as in Java 2). The model assumes a system wide *permission table* that is potentially applicable for any bundle. The *tuples* of this table consist of:

- A set of conditions
- A set of permissions

The permissions of a tuple are only applicable when *all* conditions in the set of conditions are satisfied at the time of the permission check. Certain condition types are provided by this specification, other conditions can be provided by user code. For example, the Bundle Signer Condition is satisfied when the association bundle is signed by a specific principal. If the bundle is not signed by that principal, the tuple's permissions must not apply during the check.

An instantiated condition is represented by the Condition interface, it represents an expression that can be evaluated to true or false during permission checking. The condition set must be treated as an and operation. Only if all conditions are satisfied, then the tuple is said to *match* and its permissions apply. I.e. a permission P is implied by a tuple (conditions, permissions) when

- All of its conditions are satisfied
- At least one of its permissions implies P, as defined by Java 2 security.

For example, assume the following setup for bundle A:

```
{
  [ ...BundleSignerCondition "cn=*, o=ACME, c=US" ]
```

```
    [ com.acme.Online ]
     (...AdminPermission "*", "lifecycle")
  }
```

The curly brackets {} delimit a single tuple. Conditions are delimited by square brackets [] and the permissions are delimited by parentheses (). As usual, package prefixes are left out for brevity. This syntax is used in the remainder of the document.

The example shows that both the condition org.osgi.service.condpermadmin.BundleSignerCondition must be satisfied as well as the com.acme.Online condition, before Admin Permission is granted to perform a life cycle operation on any bundle.

Binding conditional permissions to bundles is very common, though not obligatory. Therefore, a number of Condition classes are provided to define the system permissions of a bundle. This association can depend on a Bundle's:

- *Signer* – Implemented with the BundleSignerCondition class.
- *Location* – Implemented with the BundleLocationCondition class
- *Bundle Symbolic Name* – Implemented with the BundleSymbolicNameCondition class.

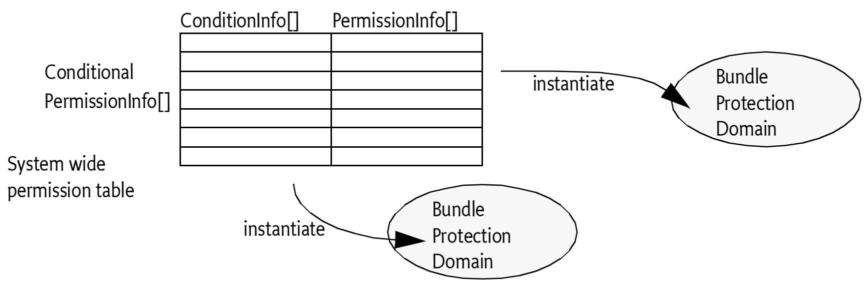
The system permissions of a bundle are dynamic and volatile, the answer to an implies method call depends therefore on the conditions that are satisfied when a permission is checked. In principle, any of the tuples in the system wide permission table can match.

8.4.1 Encoding versus Instantiation

The system wide permission table is maintained with the addConditionalPermissionInfo(ConditionInfo[],PermissionInfo[]) method. This method uses an encoded form of the conditions and the permissions. I.e. the conditions and permissions are not instantiated when in the permission table. I.e. the permission table is a dynamic *template* for the Bundle Protection Domain. It is dynamic because a Bundle Protection Domain must track the changes to the permission table immediately.

### What guarantees do we have with respect to the addCPIs method? Is it synchronous?

Figure 48 Instantiation of the permission table



The arguments to the addConditionalPermissionInfo method are ConditionInfo and PermissionInfo (from the org.osgi.service.permissionadmin package) objects. The purpose of these objects is to encode the Condition and Permission objects without instantiating them. The return value of the addConditionalPermission method is also returning an object that encodes the conditionsand permissions: a ConditionalPermissionInfo object.

The conditions and permissions of the permission table must be instantiated before the conditions can be checked. This instantiation can happen when a Bundle Protection Domain is created or the first time when the conditional permissions are needed because of a permission check.

Therefore, Condition objects must always belong to a single bundle protection domain and are never shared. Permission objects are context free and can be shared between bundle protection domains.

### It seems to be a requirement for the Conditions to have a protection domain as context (otherwise I do not understand the BundleXXXConditions), but the Permissions are context free, aren't they?

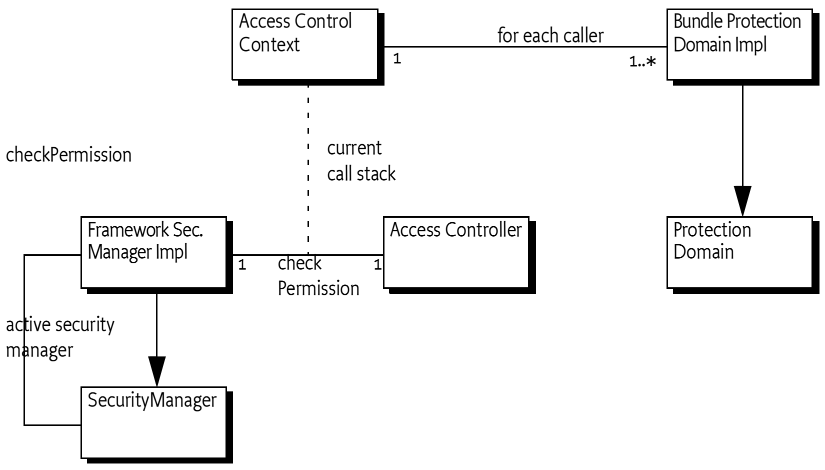
8.5 The Permission Check

The Java 2 security model has both a Security Manager and an Access Controller to perform a permission check. The core functionality is located in the AccessController and the AccessControlContext classes that cooperate with ProtectionDomain objects and Permission objects to detect if a permission is granted. ProtectionDomain objects hold the permissions for a number of classes. In the OSGi Framework, a bundle must have a single Bundle Protection Domain.

The Access Controller provides the full functionality for checking a permission. However, for backward compatibility, all system checks are tunneled through the SecurityManager checkPermission methods. The Security Manager can be replaced by a custom implementation, unlike the Access Controller (it is a final class). This model is depicted in Figure 49

The Conditional Permission Admin guards permissions with a set of conditions. I.e. a tuple's permission must only be checked if its conditions are all satisfied. Conditions can require user interaction and cause other side effects. One of the consequences of this processing model is that the execution of conditions must be highly optimized. In practice, this requires that the Framework must take the responsibility of the complete permission check. This mandates that the Framework must replace the Security Manager to be compliant.

*Figure 49		Java 2 Permission checking in OSGi bundles*



8.5.1 Check Permission Algorithm

A permission check starts when the Security Manager `checkPermission` method is called with permission `P` as argument.

`P` must be implied by the local permissions. If this is not the case, the check must end with a failure. Local permissions are described in *Local Permissions* on page 202 and *Bundle Permission Resource* on page 230.

If `P` is implied by the local permissions, the Conditional Permission Admin must get the Access Control Context in effect. It must call the `AccessController getContext()` method if it is not passed one. I.e. a specific `AccessControlContext` object can be passed as argument.

The `AccessControlContext checkPermission` method must then be called, which causes the call stack to be traversed. At each stack level the Bundle Protection Domain of the calling class is evaluated for the permission `P` using the `ProtectionDomain implies` method. This complete evaluation must take place on the same thread.

The Bundle Protection Domain must now decide which tuples in its instantiated permission table are applicable and imply `P`.

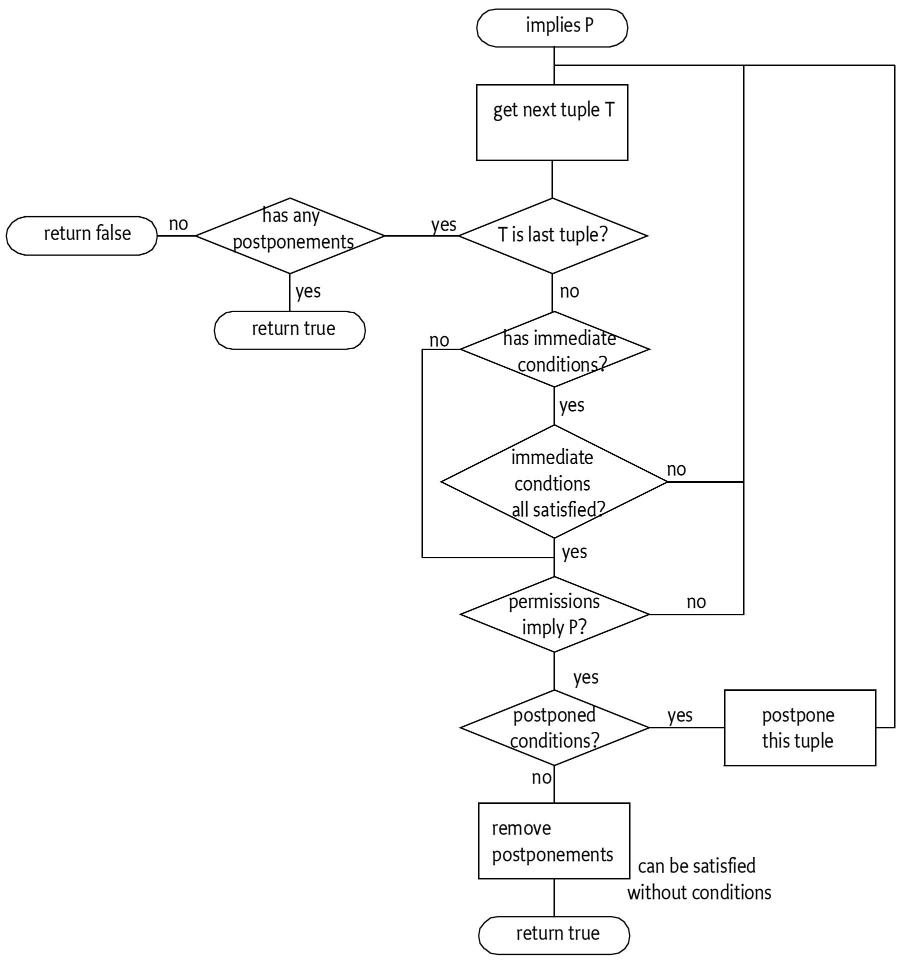
It must therefor execute the following instructions:

- For each tuple `T` in the instantiated permission table:
  - If `T` has immediate conditions, evaluate all these immediate `Condition` objects. If any of these objects is not satisfied, continue with next tuple.
  - If `T`'s permission do no imply `P`, continue with the next tuple.
  - If `T` contains postponed `Condition` objects then postpone the evaluation of `T` and continue with the next tuple.
  - Otherwise, remove any postponements and return `true`

- After all tuples have been processed
  - If there were any postponements return `true`. Otherwise, return `false`.

This algorithm is visualized in a flow chart in Figure 50.

Figure 50 *Flow chart for Bundle Protection Domain implies method*

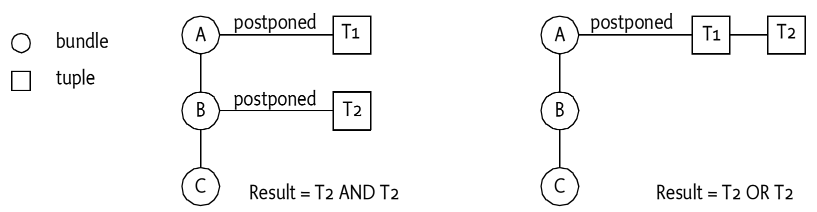


If the AccessControlContext checkPermission method returns true, there could still be a set of postponed tuples. Each of these tuples already imply permission P, otherwise they must not have been placed on the postponed list. However their Condition objects still need to be satisfied before true can be returned.

The list of postponed tuples is not a linear list. A number of Bundle Protection Domains can have contributed to the list of postponed tuples. Each Bundle Protection Domain must have the required permissions. Therefore, the Condition Permission Admin must find at least one tuple per Bundle Protection Domain it can satisfy before it can return true from the checkPermission method.

For example, if bundle A contributed T1 and bundle B contributed T2 to the postponed list, then both T1 and T2 must be satisfied. However, if only bundle A contributed T1 and T2, then either a satisfied T1 or T2 is sufficient to return true. This example is depicted in Figure 51.

Figure 51 *Evaluation of postponed tuples*



Evaluating the tuples must be grouped so that Condition objects of the same implementation can be evaluated once. For example, if the user needs to be prompted it is necessary that duplicate conditions are removed and all questions are asked at once.

### Hmmm. What happens when tuple T contains 2 conditions C1 and C2 of the same type. The semantics are that *both* must be satisfied. However, the grouped evaluation seems to indicate that either one needs to be satisfied?????

The evaluation of the postponed permissions must take place per Bundle Protection domain. All postponed tuples of that protection domain must be evaluated grouped by their class. This class is must implement a *static* method isSatisfied(Condition[], Dictionary), which is called with the group of postponed Condition objects.

### What happens if they do not implement the static isSatisfied method? Do we fall-back to the normal isSatisfied?

### I think the implementation would be nicer if we just had a isSatisfied(Dictionary) method and no static method to mess with. The condition classes can shortcut the evaluation themselves. we need to have a change anyway due tot he previous problem with two postponed conditions.

The static method can use any semantics to decide if the conditions together evaluate to true or not. If the method returns false, the permission is not implied. If it returns true, the Bundle Protection Domain implies the permission P, and the next Bundle Protection Domain on the postponed list must be verified.

#### Tatatatat ... multiple versions of classes ... despicable ... What is the influence of that on this thing?

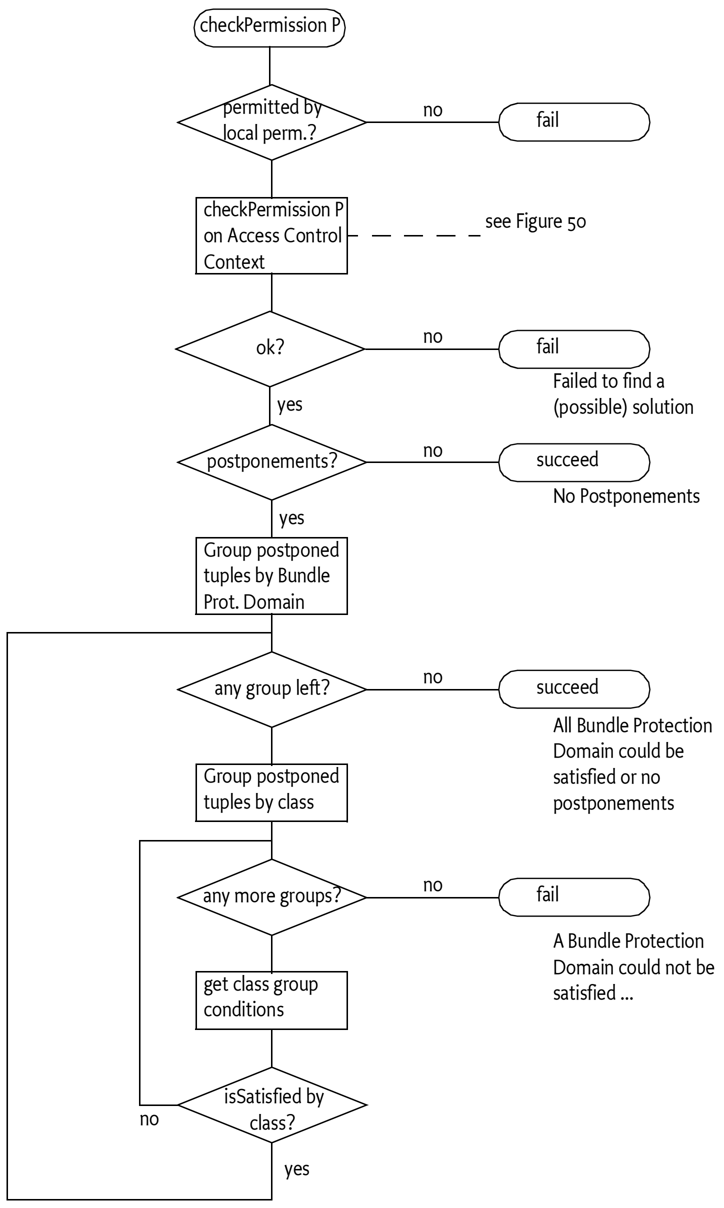
The Dictionary argument of the isSatisfied(Condition[], Dictionary) method is intended to be used by the Condition implementation class to maintain state during an invocation of the Security Manager checkPermission method. It is a Dictionary object that:

- Is specific to a Condition implementation class. I.e. different implementation classes will not share this Dictionary object.
- It is created before the isSatisfied(Condition[], Dictionary) is called for the first time.
- It is only valid during the invocation of a single checkPermission session. I.e. it is not maintained between checkPermission invocations.
- It is shared between invocations of isSatisfied(Condition[], Dictionary) method for different Bundle Protection Domains.

### Connecting the dictionary makes it hard to create cooperating conditions. Could we have the class interact with the dictionary sharing? E.g. a field in the class with a magic number or so?

The algorithm for the overal evaluation of the checkPermission method is shown in Figure 52 as a flow chart.

Figure 52 Flow chart for Security Manager checkPermission method


checkPermission P
permitted by local perm.?
no
fail
checkPermission P on Access Control Context
see Figure 50
ok?
no
fail
Failed to find a (possible) solution
yes
postponements?
no
succeed
No Postponements
yes
Group postponed tuples by Bundle Prot. Domain
any group left?
no
succeed
All Bundle Protection Domain could be satisfied or no postponements
Group postponed tuples by class
any more groups?
no
fail
A Bundle Protection Domain could not be satisfied ...
get class group conditions
isSatisfied by class?
no
yes

8.5.2 Example

A permission P is checked while bundle A, B, and C are on the call stack. Their security setup is as follows (IC = a condition that is immediately evaluated, PC is a postponed condition, P, Q, and R are permissions. The tuples are already pre-processed to only match the given bundles. I.e. all BundleXXXCondition objects have already been evaluated and removed):

```
C: {                    (Q)      }
   { [ICo]              (P)      }
   { [PC2]              (P)      }
```

First, the Bundle Protection Domain of bundle C is asked if it implies permission P. Bundle C has three tuples. The first tuple has no conditions, only a permission that does not imply permission P. The second tuple has an immediate condition IC0, which is not satisfied. Therefore, the tuple's permissions are not considered. The last tuple contains a postponed condition PC2. Permission P is implied by its permissions. It is not possible to make the decision at this moment in time, therefore the evaluation of tuple C3 is postponed. However, true is returned to indicate that the bundle is potentially permitted.

```
B: { [IC1] [PC1] [PC2]  (P)  (R)  }
   { [PC2]              (P)  (R)  }
   {                    (Q)       }
```

Next, bundle B is considered. Its first tuple has and immediate Condition object is IC1. This condition turns out to be satisfied. This tuple is a potential candidate because it has only two postponed conditions left. It is therefore necessary to check if the tuple is a possibility. It is, because its permissions imply permission P. The tuple is therefore placed on the postponed list to be checked later.

The second tuple is similar. It must also be placed on the postponed list because it implies permission P and it has a postponed condition PC2.

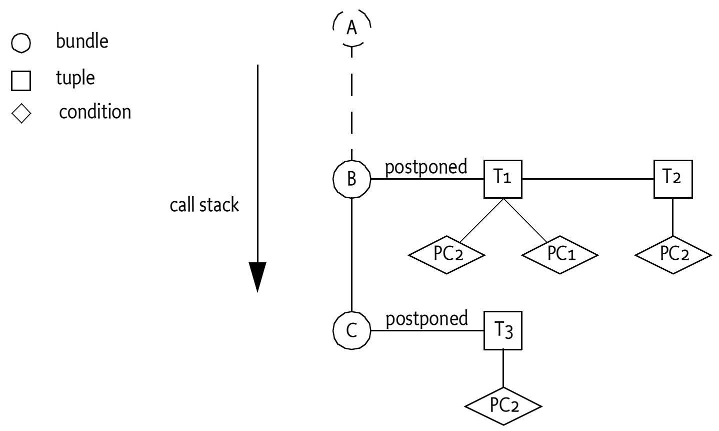
The last tuple is rejected because it does not imply permission P. However, because there are 2 tuples postponed, the bundle is potentially permitted.

```
1  A: {  [IC1]  [PC1]     (P)  (Q)  }
      {  [IC2]            (P)  (R)  }
      {                   (S)       }
```

Bundle A's IC1 is evaluated first, it is satisfied. Permission P is implied by the tuple A1's permissions, therefore this tuple is postponed for evaluation. However, the second tuple is also satisfied and it directly implies permission P. Therefore, the postponed evaluation of PC1 is removed, and bundle A is directly permitted. Tuple A3 does not have to be considered.

After the checkPermission method of the Access Control Context method returns, the Conditional Permission Admin must evaluate any postponed tuples. The list of postponed tuples looks like Figure 51.

Figure 53 *Evaluation of postponed tuples*



The order of the evaluation is not defined, but assume bundle B is evaluated first. Its postponed list contains 2 conditions of the same type: PC2. It must therefore group T1's and T2's PC2 Condition objects and call the static isSatsified method on the PC2 class. Assume that this is a user prompt for allowing connection. The PC1 class puts a marker in the given Dictionary object that the prompt was made and answered affirmative.

### Here we have the problem with multiple postponed conditions... Not sure if this is theorethical or real. For T1, the condition is PC1 & PC2. I do not see how this can be evaluated.

### The static isSatisfied method seems unnecessary if the Condition objects themselves would have an isSatisfied(Dictionary) method. Among eachother, they could maintain state, it is not exactly equal to the static method but it seems close enough? Thsi could be modelled with a ImmediateCondition and PostponedCondition interface? This would solve the problem of multiple conditions of the same type per tuple.

Bundle B is now permitted so the only one left to check is bundle C. Bundle C has a single tuple on the postponed list that has a single condition PC2. The static isSatisfied method is called again on the PC2 class. However, the Dictionary object now contains the marker from the previous invocation. The static method sees the marker and now returns true without prompting the user. This permits bundle C, and therefore the checkPermission method succeeds.

### Wow!

8.5.3 Using the Access Control Context Directly

Bundle programmers can use the standard Java API to do security checks. However, when an AccessControlContext object is used to do the check instead of the Security Manager, then the evaluation can not handle postponed conditions. Therefore, the postponed conditions must be treated as immediate conditions by the Bundle Protection Domain in this case.

8.5.4 Optimizations

Theorethically, every checkPermission method must evaluate every condition for every bundle on the call stack. I.e. the Conditional Permission Admin service must iterate through all bundles on the stack, run through the instantiated permission table of that bundle, evaluate all the conditions, test the permissions, until it finds a permission that is implied. This model would be prohibitively expensive.

The first optimization is therefore pruning the instantiated permission table. A Condition object can be pruned if it is immutable.

If an immutable Condition object is satisfied, it can be removed from the tuple's Condition objects because it can not influence the evaulation anymore. If it is not satisfied, the corresponding tuple can be completely discarded because one of the Condition objects is not satisfied, making it impossible for the tuple to be used.

If a tuple has no more Condition objects after this pruning, the evaluation of the permissions can be optimized by placing all such permissions in a single PermissionCollection object. Java 2 security has highly optimized code to do the checking of permissions when they are placed together in a Permission Collection.

For example, assume the following permission table:

```
{
   [ ...BundleLocationCondition
        "http://www.acme.com/*" ]
   ( ...SocketPermission "www.acme.com" )
} {
   [ ...BundleLocationCondition
        "http://www.et.com/*" ]
    [ ...Prompt "Call home?" ]
   ( ...SocketPermission "www.et.com" )
}
```

Assume this table is instantiated for a bundle with a location of www.acme.com/bundle.jar. The first tuple's permissions can be placed in a the special Permission Collection because the Bundle Location condition is immutable and in this case satisfied.

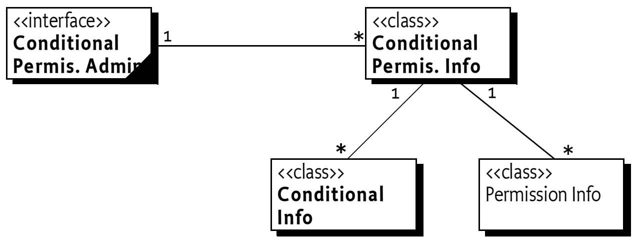
The second tuple can be discarded because it is immutable and not satisfied for the bundle's location. Any condition that is not satisfied atomically makes the tuple void.

8.6 Permission Management

The system permissions are managed with the Conditional Permission Admin service. The Conditional Permission Admin can also register the Permission Admin service, in that case the two must interact as described in *Relation to Permission Admin* on page 231.

Permissions are added in their encoded form as a tuple with the addConditionalPermissionInfo(ConditionInfo[],PermissionInfo[]) method. The method returns a ConditionalPermissionInfoobject that provides access to the tuple. These objects can also be enumerated with the getConditionalPermissionInfos() method. The ConditionalPermissionInfo delete() method allows a ConditionalPermissionInfo to be deleted. The tuple consists of an array of ConditionInfoobjects and an array of PermissionInfo objects. This is depicted in Figure 54.

Figure 54 *Structure of the Info objects.*



Both the ConditionalInfo and PermissionInfo objects can be constructed from encoded strings. The format of the encoded strings are:

```
ConditionalInfo ::= '[' qname ( quoted ) * ']'
PermissionInfo  ::= '(' qname quoted quoted ')'
```

### I think the whitespace restrictions on the grammar in the RFC are too draconian. I think it is necessary to freely allow whitespaces. Trying to find out why your permission file does not work because you have an extra whitespace is hell ... you often just do not see it.

### Should we not also allow Permission Info to have multiple parameters? Looks kind of unbalanced this way.

As a convention, ConditionalPermissionInfo objects are enclosed in curly braces ('{}', \u007B, \u007D). There are no encoding and decoding methods. As an example, a code snippet the reads a stream with conditional permissions using the curly brace convention. The method parses the file line by line. Each line is trimmed for white space, and then compared to one of the controlling characters. If a matching'}' is found, the current conditions and permissions are set.

```
   ConditionInfo   EMPTY_CS = new ConditionInfo[0];
ConditionInfo   EMPTY_PS = new PermissionInfo[0];

public void setConditionalPermissionInfo(
   ConditionalPermissionAdmin admin,
   InputStream in) throws Exception {

   Vector conditions = null;
   Vector permissions = null;

   InputStreamReader ir = new InputStreamReader(
      in, "UTF-8");
```

```
        BufferedReader br = new BufferedReader(ir);

        String line = br.readLine();

        while (line != null) {
           line = line.trim();
           switch (line.charAt(0)) {
              case '[' :
                 conditions.add(new ConditionInfo(line));
                 break;

              case '(' :
                 conditions.add(new PermissionInfo(line));
                 break;

              case '#':
                 break;

              case '{':
                 conditions = new Vector();
                 permissions = new Vector();
                 break;

              case '}' :
                 admin.addConditionalPermissionInfo(
                       (ConditionInfo[])
                          conditions.toArray(EMPTY_CS),
                       (PermissionInfo[])
                          permissions.toArray(EMPTY_PS));
                 conditions = permissions = null;
                 break;

              default:
                 throw new RuntimeException("Invalid format");

           }
           line = br.readLine();
        }
     }
```

8.7 Conditions

The purpose of a condition is to decide if a permission set is applicable or not, i.e. it acts as a guard. The condition must therefore be evaluated when a Permission object is checked against the effective permissions of a bundle.

The state of a Condition object can be obtained with its isSatisfied() method. A condition that returns true to this method is called *satisfied*. If the method throws an Exception, this should be logged and treated as if the condition is not satisfied.

Certain Condition objects could optimize their evaluations if they are activated multiple times in the same permission check. E.g. a user prompt could appear several times in a permission check but the prompt should only be asked once to the user. These conditions are called *postponed conditions*, conditions that can be verified immediately are called *immediate conditions*. The isEvaluated() method can inform if the condition is immediate or postponed. If it returns true, the isSatisfied method is quick and can be called during the permission check, otherwise the calling of isSatisfied must be postponed until the end of the check.

### Can we rename isEvaluated to isImmediate or isPostponed? I struggled with this name a lot.

For example, a condition could verify that a mobile phone is roaming, i.e. it is not in the home PLMN. This information is readily available in memory and therefore the isEvaluated() method could always return true. Alternatively, a Condition object that gets an authorization over the network should only be evaluated at most once during a permission check. Such a Condition object should return false for the isEvaluated method so all the Condition objects are evaluated together at the end of the permission check.

Condition objects only need to be evaluated multiple times when the answer can change. A Condition object that can vary its satisfiability is called *mutable*, it can be checked with the isMutable() method. If the condition is immutable, the Condtional Permission Admin can make significant optimizations by pruning tuples from its instantiated permission table. For example, the Bundle Protection Domain can prune any tuple from its view of the permission table that contains a Condition object that is immutable and not satisfied. Such conditions can never be satisfied in the future and therefore the permission tuple can be completely discarded for that Bundle Protection Domain.

This significant optimization is leveraged by the provided BundleLocationCondition, BundleSignerCondition, and BundleSymbolicNameCondition classes. The Protection Domain will never have to consider conditional permissions that do not match the protection domain's bundle. However, a Condition object can also start as a mutable condition and later become immutable. For example, a user prompt could have the following states:

- *Prompt* – The user must be prompted to get the answer, the Conditional Permission Admin will evaluate the answer to detect if it is satisfied.
- *Blanket* – The user, during an earlier prompt, has indicated it approves or denies access for that remainder of the life time of the bundle. In this state, the Condition object has become immutable.

This specification provides a number of condition classes to bind permission sets to specific bundles. However, custom code can also provide conditions. The following sections first describe how to create a condition class and then define the standard Condition classes.

8.7.1 Custom Conditions

Condition objects are constructed from ConditionInfo objects when the permission table is instantiated for a Bundle Protection Domain. The ConditionInfo object supports a variable number of arguments.

The Conditional Permission Admin must use reflection to find a public static getInstance method on the Condition implementation class that takes a Bundle object and the given number of String objects as arguments. It must return a Condition object. The class of the returned Condition object should be of the given class, otherwise the grouped evaluation does not work.

### Why can't we allow private methods for getInstance and the constructor?

However, this does not have to be a new object. I.e. the getInstance method can reuse objects if it so desires. For example, a Bundle Location Condition is immutable, it can therefore maintain only 2 instances: One for bundles that match the given location and one for the others. In the getInstance method it can compare the bundle's location with argument and return either instance.

### This looks like a highly desirable, and therfore common, pattern. Wonder if we should have two constants in the API that indicate this. I..e. Condition.FALSE, and Condition.TRUE that contain instances. This would allow the CPA to optimize compares == instead of equals? Or return null when it is not satisfied?

### Why does it not take a ConditionInfo as parameter? Simplifies and might improve memory consumption?

If no getInstance method cannot be found, the Condition Permission Admin must try to find a public constructor that takes a Bundle object and zero or more String arguments. E.g. if the ConditionInfo object looks like:

```
[ com.acme.AcmeCondition "daffy" "bugs" ]
```

The the Condition Permission Admin must look for:

```
public Condition com.acme.AcmeCondition.getInstance(
        Bundle, String, String )
public com.acme.AcmeCondition( Bundle, String, String )
```

A Condition object will be unique to a Bundle Protection Domain as explained in *Encoding versus Instantiation* on page 210. Thus, any queries made on a Condition object will be with the given Bundle object as context.

If the Conditional Permission Admin can not find a proper way to construct the Condition object, it must ###

### yes, what should it do when it cannot construct the condition? The detection happens at a very bad time ...

The next aspect that needs to be addressed is the time of evaluation and the mutability.

The cheapest Condition objects are immutable, they have almost no overhead. If a Condition object is immutable directly after it is created, then the Conditional Permission Admin can immediately shortcut future evaluations. I.e. if an immutable Condition object is not satisfied, its parent tuple can be immediately discarded, it is not even necessary to instantiate any further Condition or Permission objects.

Mutable Condition objects must be evaluated during a permission check. Permission checks are common and the evaluation of a permission should therefore be highly optimized. Additionally, care should be taken to not create additional permission checks. I.e. a mutable condition is system code, it must be designed to work in a constrained environment. The isSatisfied() method should be designed to quickly return. It should normally base its decision on variables and limit its side effects.

However, side effects are sometimes necessary, a key example is user prompting. As discussed in *Check Permission Algorithm* on page 212, the evaluation of the isSatisfied() method can be postponed towards the end of the check. The Component object must return false for the isEvaluated() method to be postponed. Postponed Condition objects must always return true for isEvaluated().

Postponed Condition objects must optimize their evaluation by implementing a static method isSatisfied(Condition[],Dictionary). This method can evaluate a number of conditions together. This grouped evaluation can be used to minimize the number of user prompting, minimize network access, etc.

### What happens if they do not implement the isSatisfied static method?

The following is the code for a condition that verifies that an action is granted by a network server. This is a postponed condition that groups all requests before it asks the host. The network code is abstracted in a Host class that is not shown here.

```
public class HostCondition implements Condition {
String       action;

public HostCondition( Bundle, String action ) {
   this.action = action;
}

public boolean isEvaluated() { return false; }
public boolean isImmutable() { return false; }
public boolean isEvaluated() { return false; }

static Host host = new Host();

static synchronized boolean isEvaluated(
   Condition[] conditions, Dictionary state ) {
   Set   granted = (Set) state.get("granted");
   if ( granted == null ) {
      granted = new TreeSet();
      state.put("granted", granted );
   }
```

```
        Set   pending = new TreeSet();
        for ( int i=0; i<conditions.length; i++ ) {
           String a = (HostCondition)conditions[i]).action;
           if ( ! granted.contains(a) )
              pending.add( a );
        }
        if ( pending.isEmpty() )
           return true;

        if ( ! host.permits( pending ) )
           return false;

        granted.addAll( pending );
        return true;
     }
  }
```

TheHostCondition has the following Condition Info representation:

```
[ HostCondition "payment" ]
```

The majority of the code is in the static isSatisfied method. The constructor only stores the action.

The static isSatisfied method first gets the set of granted permissions. The first time the method is called, this set does not exist. It is then created and stored in the state dictionary for use in later invocations.

Next, a temporary set pending is created to hold all the actions of the conditions that are checked, minus any conditions that were already granted during this invocation of the Security Manager checkPermission method. If the pending list turns out to be empty because all actions were already granted, the method returns true. Otherwise it asks the host. If the host allows the actions, the pending actions are added to the granted set in the state dictionary.

8.7.2 Threading

A Condition implementation is guaranteed that a all evalutions necessary for a single checkPermission invocation are carried out on the same thread.

8.7.3 Condition Life Cycle

Condition objects will get instantiated when the framework is restarted or the Bundle Protection Domain is created. Framework implementations can also use optimizations that cause Condition objects to get created and destroyed multiple times within the lifetime of an instance of a Bundle Protection Domain. I.e. an implementation of a Condition class must not make any assumptions about its creation or dereferencing.

### should or must?

8.7.4 Context Dependent Conditions

### I want to remove this section. Any silly person can do this if they want to hang themselves, we do not have to standardize the hanging process :-)

In certain situations, the evaluation of a Condition object requires information specific to the invocation. I.e. information that is not available to the condition evaluation unless it is specifically given to it as an argument. For example, an installation is only permitted to take place when the cost of the channel used to transfer the bundle is low.

The permission check is done on the same thread as the call of the SecurityManager checkPermission method. This makes it possible that the programmer uses a thread local variable to store the context for the specific Condition class. After the check is done this thread local variable must be reset. This mechanism is obviously error prone and must be used with utmost care. Additionally, the mechanism is difficult to protect against malicious code. It is therefore usually better to use Condition objects that do not require such a context.

### This is so ugly ... and error prone. And I have a hard time finding a good use cases. The transfer cost does not really seem to make sense. It is not very likely that you know the type of channel as well as handle the install operations.

This is illustrated with an example of a Transfer Cost Condition skeleton:

```
1   public class TransferCostCondition implements Condition {
    int maxCost;

    public TransferCostCondition(String cost) {
       maxCost = Integer.parseInt(cost);
    }

    static ThreadLocal context = new ThreadLocal();

    public static void setTransferCost(int cost) {
       context.set(new Integer(cost));
    }

    public static void resetTransferCost() {
       context.set(null);
    }

    public boolean isSatisfied() {
       Integer i = (Integer)context.get();
       return i!=null ? i.intValue()<=maxCost : false;
    }
  }
```

This Condition is constructed with a maximum transfer cost. It is considered satisfied when the context transfer cost is less that the maximum transfer cost. Since there the state of the Transfer Cost Condition is fully contained in the class it is an immediate Condition. When the transfer cost is known, it should be set using the static setTransferCost and resetTransferCost methods.

```
1   TransferCostCondition.setTranferCost(2);
  // … Other Processing …
  sm.checkPermission(
    new SocketPermission("www.ibm.com:80", "connect"));
```

```
// … Other Processing …
TransferCostCondition.resetTransferCost();
```

If the permission table for the calling bundle would look like:

```
{
   [org.osg.meg.TransferCost 3]
   (java.net.SocketPermission "*" "connect")
}
```

Then, the snippet runs successfully. It would fail if the caller had the following setup:

```
{
   [org.osg.meg.TransferCost 1]
   (java.net.SocketPermission "*" "connect")
}
```

Context dependent conditions are very similar, but not equal, to permissions. A Transfer Cost Permission could easily check a transfer cost because permissions have arguments. The subtle difference between the two cases is that a Condition object can guard a number of permissions at the same time.

### I feel very uneasy with the example, so maybe the last part should go. But I feel the example is very contrived and below the standard we have in the OSGi so far. However, I cannot think of another one.

8.7.5 Recursive Evaluations

If there is a chance that permissions will be checked in code being called by isSatisfied, the implementer of the Condition should use the AccessController doPrivileged to ensure needed permissions.

Implementers must handle recursive permission checks. Implementers of Permission classes have the same issue, but Condition classes in general have much more sophisticated logic than Permissions classes. For example, a User Prompt Condition has the potential to cause many permission checks as it interacts with the UI.

There is always the possibility that a chain of method invocations can cause the same Condition object to get re-evaluated. Since the framework cannot detect all such recursive calls, it is up to the implementer of the Condition class to detect a recursive call to a Condition object and take action to avoid infinite recursion.

The Framework must detect a secondary evaluation of a Condition object when it occurs on the same thread. In that case, the second evaluation must be treated as returning false. For example, if a User Prompt condition is evaluated and this evaluation accesses the UI, which in its turn checks a permission that causes the evaluation of a User Prompt Condition, then this second evaluation must not take place and be treated as a false.

### Do we need to log this?

### This is only true for the instance is it? Not for the class. I.e. if two different user prompts occur, they can be recursive? I guess we should leave it completely up to the implementers?

8.8 Standard Conditions

This specification provides a number of standard conditions. The osgi.jar file that contains all the specification classes, does contain a non-functional implementation of these conditions. Actual Framework implementers must replace these classes with an implementation that is tied to their Framework implementation for efficiency reasons.

### Yack! Yack! Yack!!!!!!!!! If we would have a Certificate repository we could at least have some implementation. This is creating so many problems in the build ...

8.8.1 Bundle Signer Condition

A Bundle Signer Condition is satisfied when the related bundle is signed with a certificate that matches its argument. I.e. this condition can be used to assign permissions to bundles that are signed by certain principals.

The Bundle Signer Condition must be created through its static getInstance(Bundle,String) method. The string argument is a matching Distinguished Name as defined in *Certificate Matching* on page 24. For example:

```
[ ...BundleSignerCondition "* ;cn=S&V, o=Tweety Inc., c=US"]
```

The Bundle Signer Condition is immutable and can be completely evaluated during the getInstance method.

8.8.2 Bundle Location Condition

The Bundle Location Condition matches its argument against the location string of the bundle argument. Bundle location matching provides many of the advantages of signing without the overhead. However, using locations as the authenticator requires that the download locations are secured and cannot be spoofed. For example, an Operator could permit Enterprises by forcing them to download their bundles from specific locations. To make this reasonable secure, at least the HTTPS protocol should be used. The Operator can then use the location to assign permissions.

```
https://www.acme.com/download/-       Apps from ACME
https://www.operator.com/download/*   Operator apps
```

The Bundle Location Condition must be created through its static getInstance(Bundle,String) method. The string argument is a location string with possible wildcards ('*'). Wildcards are matched identical to the matching of File Permissions. I.e. it is allowed to end the string with /* or /-. The /* indicates that the location can have any further characters after the / except for another slash. The /- form indicates that further levels with extra slashes are allowed, i.e. all subdirectories. For example:

```
http://www.acme.com/*
ftps://www.acme.com/-
http://www.acme.com
```

### This syntax must become equal to to the getEntries syntax @ Bundle

The Bundle Location Condition is satisfied when its argument can be matched with the actual location.

The Bundle Location Condition is immutable and can be completely evaluated during the getInstance method.

8.8.3 Bundle Symbolic Name Condition

### The more I think about this condition, the more I feel we should not have these name based conditons and permissions

### Don't we have a race condition here? A bundle requires a BSN Permission, but we cannot evaluate if we have that permissions until the conditions are satisfied. I.e. it looks like a BSN Condition is not usable until after a permission check?

The Bundle Symbolic Name is required to be a globally unique name. This opens the possibility of spoofing. Bundle Symbolic Names must therefore be protected with a Bundle Permission. Using this permission, the Operator can enforce a naming structure on the Bundle Symbolic Name. For example, it could require that each Enterprise that downloads applications uses a prefix. I.e. ACME would be forced to use com.acme.* as prefix, legal names would be com.acme.games.tetris.

If such a name space is enforced, it is also possible to use the naming structure with conditions. However, to use the Bundle Symbolic Name securely, the bundle first needs to be granted permission to use that Bundle Symbolic Name. This BundlePermission[<bsn>,PROVIDE] must be guarded by another mechanism than Bundle Symbolic Name, for example Bundle Signer Condition or Bundle Location Condition.

For a concrete example:

```
{
   [ org....condpermadmin.BundleLocationPermission
        "https://www.acme.com/download/*" ]
   ( org....framework.BundlePermission
      "com.acme.*" "provide" )
}
{
    [ org...condpermadmin.BundleSymbolicNameCondition
        "com.acme.*" ]
   ( ... other permissions )
}
```

Great care should be take to prevent assigning permissions to a bundle with a Bundle Symbolic Name Condition if that bundle does not have the permission to carry that symbolic name.

### This sucker is useless.Why would one have to go through this indirection? It seems that a BSN Condition always requires another guard to function securely. They it seems better to ALWAYS use that other guard. QED?

### If you want to keep it, we need to layout the rules how and when the BundlePermission interacts with the BSN Condition ...

The Bundle Symbolic Name Condition must be created through its static getInstance(Bundle,String,String) or getInstance(Bundle,String) method. The string argument is a Bundle Symbolic Name. The string can use the same wildcards as the OSGi filter. I.e. wildcards must match any number of characters but can appear multiple times.

The second argument is the version range as defined in the Framework specification *Bundle-ManifestVersion* on page 39

If the second parameter is not set, the Bundle Symbolic Name Condition must be satisfied by any version of a bundle with a Bundle Symbolic Name that matches the first argument.

The Bundle Location Condition is immutable and can be completely evaluated during the getInstance method.

8.9 Bundle Permission Resource

### Could we move this to the framework security layer?

Bundles can convey their local permissions using the file META-INF/permissions.perm. This must be a UTF-8 encoded file.

### I guess we move the permission file to OSGI-INF?

The format of the file is line based, lines are not limited in length but must be ended with a carriage return, line feed pair ('\n\r' \u000D, \u000A):

```
permission.perm ::= line *
line            ::= ( comment | pinfo ) '\r\n'
comment         ::= ( '#' | '//' |  /* blank */ )
pinfo           ::= '(' qname argument argument ')'
```

### Is there any reason we do not support conditions in the local permissions?

Each permission must be listed on its own line using the encoded form of Permission Info. Comment lines are allowed. They consist of lines starting with a #or //, where leading spaces must be ignored. Multiple white spaces outside quotes must be treated as a single white space.

For example (package prefixes left out for brevity, i.e. ... must be replaced with the appropriate prefix.):

```
# Friday, Feb 24 2005
# ACME, chess game
( ..ServicePermission   "..log.LogService" "GET" )
( ..PackagePermission   "..log" "IMPORT" )
( ..ServicePermission   "..cm.ManagedService" "REGISTER" )
( ..PackagePermission   "..cm" "IMPORT" )
( ..ServicePermission   "..useradmin.UserAdmin" "GET" )
( ..PackagePermission   "..cm" "SET" )
( ..ConfigurationPermission "com.acme.*" "GET" )
( ..PackagePermission   "com.acme.chess", "IMPORT,EXPORT" )
( ..PackagePermission   "com.acme.score", "IMPORT" )
```

### Could we not implicitly get permission for all manifest information? All these puny permissions are so error prone and redundant? The signer can inspect the manifest as well as this file, so there is no difference in control/security.

If this resource is present in the Bundle JAR, it will set the local permissions. If it is not present, the local permissions must be All Permission.

8.9.1 Removing the Bundle Permission Resource

An attacker could circumvent the local permission by simply removing the permissions.perm file from the bundle. This would remove any local permissions that were required by a signer of the bundle. To prevent this type of attack the Conditional Permission Admin must detect that the permissions.perm resource was signed (i.e. present in the manifest) but that it is not in the JAR. If the bundle is being installed when this condition is detected, the install must fail with a Bundle Exception.

### Must mention this in the install method

8.10 Relation to Permission Admin

If the framework provides both a Conditional Permission Admin service and a Permission Admin service then the location bound permission of Permission Admin service override any information of the Conditional Permission Admin service. Otherwise, the concepts of Conditional Permission Admin service apply.

The Permission Admin defines a concept of *Default Permissions*, which is not explicitly supported by Conditional Permission Admin. Default permissions are now modeled with an empty set of conditions. An empty set of conditions apply to all bundles so there is no need for a special case.

New applications should use the Conditional Permission Admin service. The Permission Admin service will be deprecated in a future release.

### How is Bundle.hasPermission() handled?

### I would like to express Permission Admin as a BundleLocationCondition instead of treating it differently

8.11 Security

8.11.1 Service Registry Security

8.11.1.1 Conditional Permission Admin Service

The Conditional Permission Admin service should be part of the Framework and therefore has All Permission.

8.11.1.2 Client

```
ServicePermission    ..ConditionalPermissionAdmin  GET
PackagePermission    ..condpermadmin               IMPORT
AdminPermission      <scope>                       PERMISSION
```

The <scope> of the client depends on what bundles must be managed. If this is any bundle, the <scope> must be *. Otherwise it must select the signer or location of the target bundles as described in *AdminPermission* on page 99.

### RFC 73 says that the <scope> must be the system bundle? I am actually very confused now ... I cannot find the restriction that you could never set a permission you did not have? We did have that one day? Did we kill it?

8.12 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
```

8.12.1 Summary

- *BundleLocationCondition* - Checks to see if a Bundle matches the given location pattern. [p.232]
- *BundleSymbolicNameCondition* - This Condition evalutes the symbolic name and version of a bundle. [p.233]
- *Condition* - This interface is used to implement Conditions that are bound to Permissions using ConditionalPermissionCollection. [p.234]
- *ConditionalPermissionAdmin* - This is a framework service that allows ConditionalPermissionInfos to be added to, retrieved from, and removed from the framework. [p.235]
- *ConditionalPermissionInfo* - This interface describes a binding of a set of Conditions to a set of Permissions. [p.236]
- *ConditionInfo* - Condition representation used by the Conditional Permission Admin service. [p.236]

8.12.2 public class BundleLocationCondition implements Condition

Checks to see if a Bundle matches the given location pattern. Pattern matching is done using FilePermission style patterns.

8.12.2.1 public BundleLocationCondition(Bundle bundle, String location)

bundle the Bundle being evaluated.

location the location specification to match the Bundle location to. Matching is done according to the patterns documented in FilePermission.

□ Constructs a condition that tries to match the passed Bundle's location to the location pattern.

8.12.2.2 public boolean isEvaluated( )

□ This method is always true since this is an immutable Condition.

Returns always true.

See Also `org.osgi.service.condpermadmin.Condition.isEvaluated()` [p.234]

8.12.2.3 public boolean isMutable( )

□ This Condition never changes, so this method always returns false.

Returns always returns false.

See Also `org.osgi.service.condpermadmin.Condition.isMutable()` [p.235]

8.12.2.4 public boolean isSatisfied( )

□ This method returns true if the location of the bundle matches the the location pattern that was used to construct this Condition. The matching is done according to the matching scheme in FilePermission.

Returns true if the location of the bundle matches the location pattern of this condition.

See Also `org.osgi.service.condpermadmin.Condition.isSatisfied()` [p.235], `java.io.FilePermission`

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

conds the conditions to check for satisfiability.

context not used.

□ This method should never get called since this is an immutable Condition. As implemented it simply loops through all the Conditions calling isSatisfied()

Returns true if all of the conditions are satisfied.

See Also

`org.osgi.service.condpermadmin.Condition.isSatisfied(org.osgi.service.condpermadmin.Condition[])` [p.234]

8.12.3 public class BundleSymbolicNameCondition implements Condition

This Condition evalutes the symbolic name and version of a bundle. There are two constructors. One takes a BundleSymbolicName that may have`*' to do wildcarding as defined in * LDAP queries ?[5]. The second constructor adds is the version range as defined in section 5.4 of RFC 79. This condition is immutable since the bundle symbolic name cannot change after the bundle has been installed.

8.12.3.1 public BundleSymbolicNameCondition(Bundle bundle, String nameExp) throws InvalidSyntaxException

bundle the bundle to be checked.

nameExp the expression to check against the bundle symbolic name. The expression is defined in ITEF RFC 2253. Any special characters must be escaped as specified in the RFC.

□ Checks the symbolic name of the given bundle against the given expression.

Throws `InvalidSyntaxException` –

8.12.3.2 public BundleSymbolicNameCondition(Bundle bundle, String nameExp,

String versionExp) throws InvalidSyntaxException

bundle the bundle to be checked.

nameExp the expression to check against the bundle symbolic name. The expression is defined in ITEF RFC 2253. Any special characters must be escaped as specified in the RFC.

versionExp specifies the version range of the bundle that will satisfy this condition.

□ Checks the symbolic name and the version of the given bundle against the given expressions.

Throws `InvalidSyntaxException` –

8.12.3.3 public boolean isEvaluated( )

□ This always returns true since this is an immutable condition.

Returns always true.

See Also `org.osgi.service.condpermadmin.Condition.isEvaluated()` [p.234]

8.12.3.4 public boolean isMutable( )

□ This method always returns false since this is an immutable condition.

Returns always false.

See Also `org.osgi.service.condpermadmin.Condition.isMutable()` [p.235]

8.12.3.5 public boolean isSatisfied( )

□ Returns true if the Bundle that was used to construct this conditions matches the SymbolicName description.

Returns true if the Bundle that was used to construct this conditions matches the SymbolicName description.

See Also `org.osgi.service.condpermadmin.Condition.isSatisfied()` [p.235]

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

conds the conditions to be evaluated.

context not used.

□ This method really shouldn't be called since this is an immutable condition. It simply calls isSatisifed() on each member of conds and returns true if they all return true.

Returns

See Also

`org.osgi.service.condpermadmin.Condition.isSatisfied(org.osgi.service.condpermadmin.Condition[], java.util.Dictionary)` [p.235]

8.12.4 public interface Condition

This interface is used to implement Conditions that are bound to Permissions using ConditionalPermissionCollection. The Permissions of the ConditionalPermissionCollection can only be used if the associated Condition is satisfied.

8.12.4.1 public boolean isEvaluated( )

- This method returns true if the Condition has already been evaluated, and its satisfiability can be determined from its internal state. In other words, isSatisfied() will return very quickly since no external sources, such as users, need to be consulted.

8.12.4.2 public boolean isMutable( )

- This method returns true if the satisfiability may change.

8.12.4.3 public boolean isSatisfied( )

- This method returns true if the Condition is satisfied.

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

conds the array of Conditions that must be satisfied

context a Dictionary object that implementors can use to track state. If this method is invoked multiple times in the same permission evaluation, the same Dictionary will be passed multiple times. The SecurityManager treats this Dictionary as an opaque object simply creates an empty dictionary and passes it to subsequent invocations if multiple invocatios are needed.

- This method returns true if the set of Conditions are satisfied. Although this method is not static, it should be implemented as if it were static. All of the passed Conditions will have the same type and will correspond to the class type of the object on which this method is invoked.

Returns true if all the Conditions are satisfied.

8.12.5 public interface ConditionalPermissionAdmin

This is a framework service that allows ConditionalPermissionInfos to be added to, retrieved from, and removed from the framework.

8.12.5.1 public ConditionalPermissionInfo addConditionalPermissionInfo(ConditionInfo[] conds, PermissionInfo[] perms)

conds the Conditions that need to be satisfied to enable the corresponding Permissions.

perms the Permissions that are enable when the corresponding Conditions are satisfied.

- This is a framework service that allows ConditionalPermissionInfos to be added to, retrieved from, and removed from the framework.

Returns the ConditionalPermissionInfo that for the newly added Conditions and Permissions.

8.12.5.2 public AccessControlContext getAccessControlContext(String[] signers)

signers the signers that will be checked agains BundleSignerCondition.

- Returns the AccessControlContext that corresponds to the given signers.

Returns an AccessControlContext that has the Permissions associated with the signer.

8.12.5.3 public Enumeration getConditionalPermissionInfos( )

☐ Returns the ConditionalPermissionInfos that are currently managed by ConditionalPermissionAdmin. The Enumeration is made up of ConditionalPermissionInfos. Calling ConditionalPermissionInfo.delete() will remove the ConditionalPermissionInfo from ConditionalPermissionAdmin.

Returns the ConditionalPermissionInfos that are currently managed by ConditionalPermissionAdmin. The Enumeration is made up of ConditionalPermissionInfos.

8.12.6 public interface ConditionalPermissionInfo

This interface describes a binding of a set of Conditions to a set of Permissions. Instances of this interface are obtained from the ConditionalPermissionAdmin service. This interface is also used to remove ConditionalPermissionCollections from ConditionPermissionAdmin.

8.12.6.1 public void delete( )

☐ Removes the ConditionalPermissionCollection from the ConditionalPermissionAdmin.

8.12.6.2 public ConditionInfo[] getConditionInfos( )

☐ Returns the ConditionInfos for the Conditions that must be satisfied to enable this ConditionalPermissionCollection.

8.12.6.3 public PermissionInfo[] getPermissionInfos( )

☐ Returns the PermissionInfos for the Permission in this ConditionalPermissionCollection.

8.12.7 public class ConditionInfo

Condition representation used by the Conditional Permission Admin service.

This class encapsulates two pieces of information: a Condition *type* (class name), which must implement `Condition`, and the arguments passed to its constructor.

In order for a Condition represented by a `ConditionInfo` to be instantiated and considered during a permission check, its Condition class must be available from the system classpath.

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

type The fully qualified class name of the condition represented by this `ConditionInfo`. The class must implement `Condition` and must define a constructor that takes a `Bundle` and the correct number of argument strings.

args The arguments that will be passed to the constructor of the `Conditon` class identified by `type`.

☐ Constructs a `ConditionInfo` from the given type and args.

Throws `NullPointerException` – if `type` is `null`.

8.12.7.2 **public ConditionInfo(String encodedCondition)**

encodedCondition The encoded `ConditionInfo`.

□ Constructs a `ConditionInfo` object from the given encoded `ConditionInfo` string.

Throws `IllegalArgumentException` – if `encodedCondition` is not properly formatted.

See Also `getEncoded`[p.237]

8.12.7.3 **public boolean equals(Object obj)**

obj The object to test for equality with this `ConditionInfo` object.

□ Determines the equality of two `ConditionInfo` objects. This method checks that specified object has the same type and args as this `ConditionInfo` object.

Returns `true` if `obj` is a `ConditionInfo`, and has the same type and args as this `ConditionInfo` object; `false` otherwise.

8.12.7.4 **public final String[] getArgs( )**

□ Returns arguments of this `ConditionInfo`.

Returns The arguments of this `ConditionInfo`. have a name.

8.12.7.5 **public final String getEncoded( )**

□ Returns the string encoding of this `ConditionInfo` in a form suitable for restoring this `ConditionInfo`.

The encoding format is:

```
[type "arg0" "arg1" ...]
```

where *argX* 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 "*arg0*" and between all arguments.

Returns The string encoding of this `ConditionInfo`.

8.12.7.6 **public final String getType( )**

□ Returns the fully qualified class name of the condition represented by this `ConditionInfo`.

Returns The fully qualified class name of the condition represented by this `ConditionInfo`.

8.12.7.7 **public int hashCode( )**

□ Returns the hash code value for this object.

Returns A hash code value for this object.

8.12.7.8 public String toString( )

☐ Returns the string representation of this `ConditionInfo`. The string is created by calling the `getEncoded` method on this `ConditionInfo`.

Returns The string representation of this `ConditionInfo`.

8.13 References

[56] *Java 1.X ### (which Java do we base this on?)*

9 Permission Admin Service Specification

Version 1.1

PermissionAdmin.setPermissionsThe bundle calling setPermissions cannot assign a bundle any permission that the calling bundle does not have. This is to prevent privilege elevation. This evaluation can only be done on instantiated Permissions. If some of the PermissionInfos cannot be evaluated with respect to the call stack of the caller of setPermissions and the caller does not have AllPermission, a SecurityException will be thrown.

PermissionAdmin.setDefaultPermissionsRequires AdminPermission(0,"permission"). The name value must be 0 since this method affects the entire framework.

PackageAdmin.refreshPackagesRequires AdminPermission(0,"resolve"). The name value must be 0 since this method affects the entire framework.

PackageAdmin.resolveBundlesRequires AdminPermission(0,"resolve"). The name value must be 0 since this method affects the entire framework.

StartLevel.setStartLevelRequires AdminPermission(0,"startlevel"). The name value must be 0 since this method affects the entire framework.

StartLevel.setInitialBundleStartLevelRequires AdminPermission(0, "startlevel"). The name value must be 0 since this method affects the entire framework.

5.1.4 Setting AdminPermission via PermissionAdmin

When assigning permissions to be bundle via PermissionAdmin, it is difficult for the administrator to have a priori knowledge of the bundle id that is assigned to a bundle. Since bundle ids are assigned locally by the framework during install of a bundle, a given bundle may have a unique id on every framework upon which it is installed. Thus we need a mechanism for the administrator to use to select the appropriate bundle id so that AdminPermissions can be created for a bundle's ProtectionDomain.

So in order to provide a more flexible way to specify the bundle ids for an AdminPermission, PermissionAdmin will provide special support for assigning AdminPermissions. When a PermissionInfo with an AdminPermission type is specified to PermissionAdmin, the name parameter of the PermissionInfo must be either a bundle id or a filter string. The filter string can only be used with PermissionInfo. If a bundle constructs an AdminPermission object using a filter string, it will never be implied by any other AdminPermission or AdminPermission collection.

The supported filter string keys are:

Key	Description

9.1 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 Architecture* on page 11.

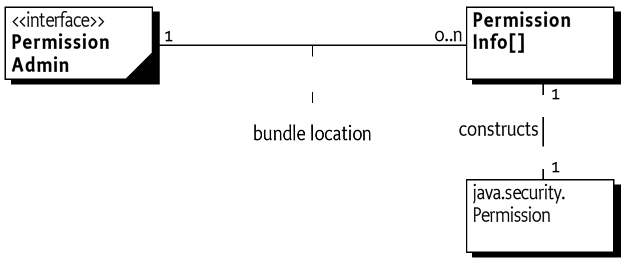
9.1.1 Essentials

- *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.

9.1.2 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 Identifiers* on page 82.

Figure 55 *Class Diagram org.osgi.service.permissionadmin.*



9.1.3 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.

9.2 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.

9.2.1 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.

9.3 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.

9.4 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.

9.5 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; version=1.1
```

9.5.1 Summary

- *PermissionAdmin* - The Permission Admin service allows management agents to manage the permissions of bundles. [p.243]
- *PermissionInfo* - Permission representation used by the Permission Admin service. [p.245]

9.5.2 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.

9.5.2.1 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.

9.5.2.2 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.

9.5.2.3 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.

9.5.2.4 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.

9.5.2.5 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.

9.5.3 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.

9.5.3.1 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.

9.5.3.2 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.246]

9.5.3.3 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.

9.5.3.4 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.

9.5.3.5 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.

9.5.3.6 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.

9.5.3.7 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.

9.5.3.8 public int hashCode( )

- Returns the hash code value for this object.

Returns A hash code value for this object.

9.5.3.9 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.

10 URL Handlers Service Specification

Version 1.0

10.1 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 97.

10.1.1 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.

10.1.2 Entities

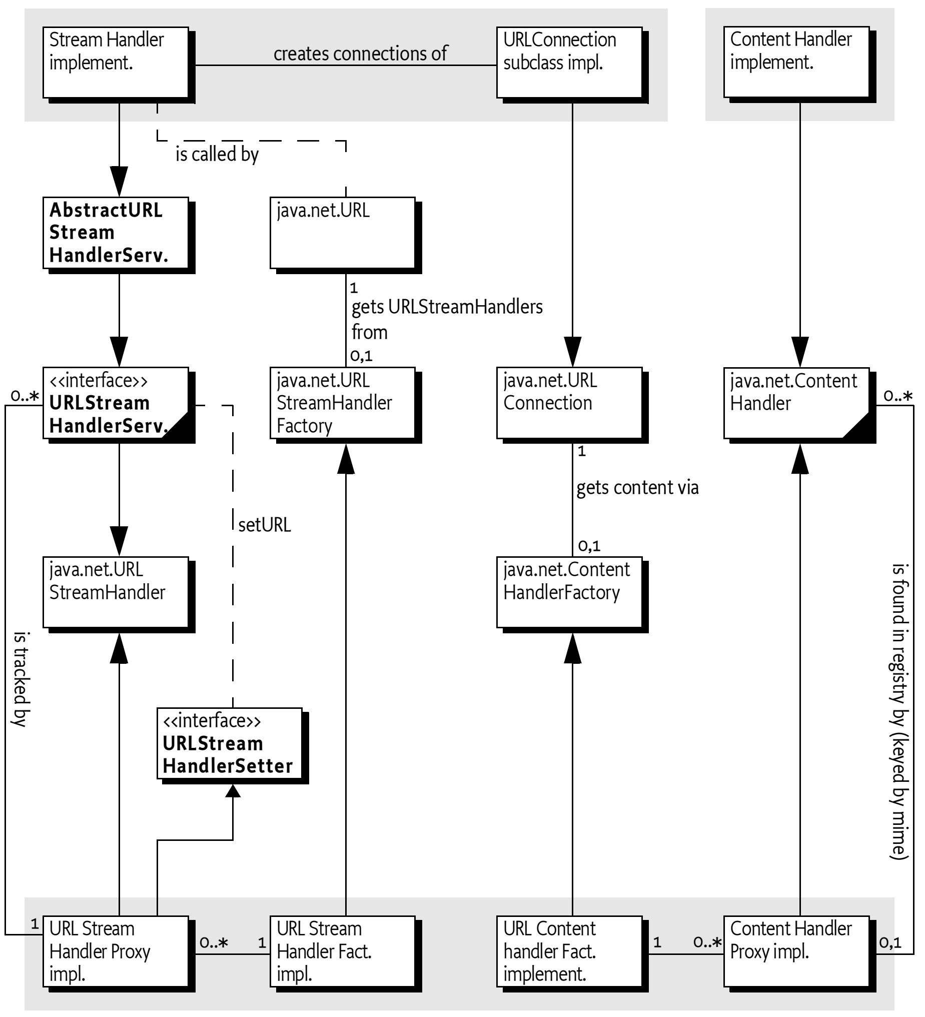
- *Scheme* – An identifier for a specific protocol. For example, "http" is a scheme for the Hyper Text Transfer Protocol. A scheme is implemented in a java.net.URLStreamHandler sub-class.
- *Content Type* – An identifier for the type of the content. Content types are usually referred to as MIME types. A content type handler is imple-

mented 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 [59] *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 56 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

10.1.3 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.

10.2 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 mechansism is described in the Javadoc for the URLStreamHandler and ContentHandler class, see [57] *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.

10.3 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 [59] *MIME Multipurpose Internet Mail Extension*.

10.3.1 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 164) 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.

10.3.2 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.

10.3.3 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
   ```

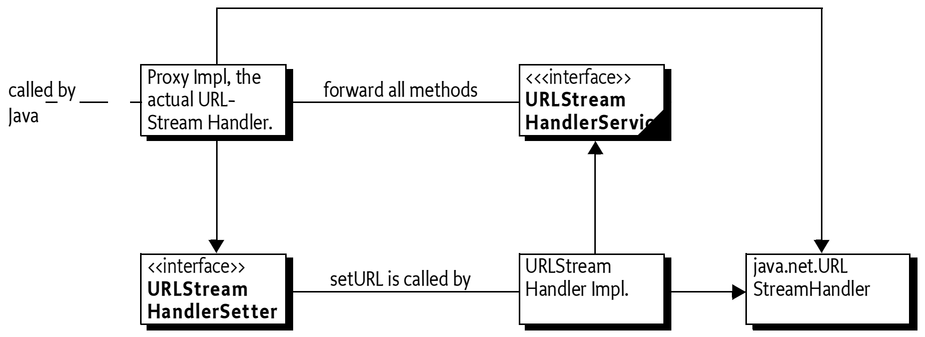
The Java run-time specific packages should be listed in the appropriate properties so that implementations of the URL Stream Handler Factory and Content Handler Factory can be made aware of these packages.

10.3.4 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 57 *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.

10.4 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";
      }
   }
```

10.5 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();
      }
   }
```

10.6 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.

10.7 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; version=1.0
```

10.7.1 Summary

- *AbstractURLStreamHandlerService* - Abstract implementation of the URLStreamHandlerService interface. [p.259]
- *URLConstants* - Defines standard names for property keys associated with URLStreamHandlerService[p.261] and java.net.ContentHandler services. [p.260]
- *URLStreamHandlerService* - Service interface with public versions of the protected java.net.URLStreamHandler methods. [p.261]
- *URLStreamHandlerSetter* - Interface used by URLStreamHandlerService objects to call the setURL method on the proxy URLStreamHandler object. [p.262]

10.7.2 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.

10.7.2.1 protected URLStreamHandlerSetter realHandler

The URLStreamHandlerSetter object passed to the parseURL method.

10.7.2.2 public AbstractURLStreamHandlerService( )

10.7.2.3 public boolean equals(URL u1, URL u2)

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

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

10.7.2.4 public int getDefaultPort( )

□ This method calls super.getDefaultPort.

See Also `java.net.URLStreamHandler.getDefaultPort`

10.7.2.5 public InetAddress getHostAddress(URL u)

- This method calls super.getHostAddress.

See Also `java.net.URLStreamHandler.getHostAddress`

10.7.2.6 public int hashCode(URL u)

- This method calls super.hashCode(URL).

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

10.7.2.7 public boolean hostsEqual(URL u1, URL u2)

- This method calls super.hostsEqual.

See Also `java.net.URLStreamHandler.hostsEqual`

10.7.2.8 public abstract URLConnection openConnection(URL u) throws IOException

See Also `java.net.URLStreamHandler.openConnection`

10.7.2.9 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`

10.7.2.10 public boolean sameFile(URL u1, URL u2)

- This method calls super.sameFile.

See Also `java.net.URLStreamHandler.sameFile`

10.7.2.11 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.

10.7.2.12 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)`

10.7.2.13 public String toExternalForm(URL u)

- This method calls super.toExternalForm.

See Also `java.net.URLStreamHandler.toExternalForm`

10.7.3 public interface URLConstants

Defines standard names for property keys associated with URLStreamHandlerService[p.261] and java.net.ContentHandler services.

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

10.7.3.1 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.

10.7.3.2 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.

10.7.4 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.262] 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.259]

10.7.4.1 public boolean equals(URL u1, URL u2)

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

10.7.4.2 public int getDefaultPort( )

See Also java.net.URLStreamHandler.getDefaultPort

10.7.4.3 public InetAddress getHostAddress(URL u)

See Also java.net.URLStreamHandler.getHostAddress

10.7.4.4 public int hashCode(URL u)

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

10.7.4.5 public boolean hostsEqual(URL u1, URL u2)

See Also java.net.URLStreamHandler.hostsEqual

10.7.4.6 public URLConnection openConnection(URL u) throws IOException

See Also java.net.URLStreamHandler.openConnection

10.7.4.7 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`

10.7.4.8 public boolean sameFile(URL u1, URL u2)

See Also `java.net.URLStreamHandler.sameFile`

10.7.4.9 public String toExternalForm(URL u)

See Also `java.net.URLStreamHandler.toExternalForm`

10.7.5 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.261] 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.

10.7.5.1 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.

10.7.5.2 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)`

10.8 References

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

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

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

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

11 Service Tracker Specification

Version 1.2

11.1 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.

11.1.1 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.

11.1.2 Operation

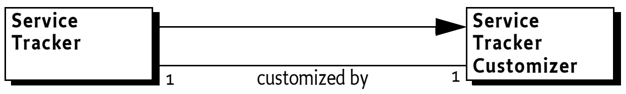
The fundamental tasks of a `ServiceTracker` object are:

- 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.

11.1.3 Entities

Figure 58 *Class diagram of org.osgi.util.tracker*



11.1.4 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.

11.2 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.

11.3 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.

11.4 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.

11.4.1 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.

11.5 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);
}
```

11.6 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.

11.7 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.

11.8 org.osgi.util.tracker

The OSGi Service Tracker 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.util.tracker; version=1.3
```

11.8.1 Summary

- *ServiceTracker* - The ServiceTracker class simplifies using services from the Framework's service registry. [p.263]
- *ServiceTrackerCustomizer* - The ServiceTrackerCustomizer interface allows a ServiceTracker object to customize the service objects that are tracked. [p.263]

11.8.2 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.

11.8.2.1 protected final BundleContext context

Bundle context this ServiceTracker object is tracking against.

11.8.2.2 protected final Filter filter

Filter specifying search criteria for the services to track.

Since 1.1

11.8.2.3 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.

11.8.2.4 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.

11.8.2.5 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

11.8.2.6 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.263]

11.8.2.7 public synchronized void close( )

□ Close this ServiceTracker object.

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

11.8.2.8 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.

11.8.2.9 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.

11.8.2.10 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

11.8.2.11 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.

11.8.2.12 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.

11.8.2.13 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

11.8.2.14 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.263]

11.8.2.15 public void open( )

- Open this ServiceTracker object and begin tracking services.

This method calls open(false).

Throws `IllegalStateException` – if the BundleContext object with which this ServiceTracker object was created is no longer valid.

See Also `open(boolean)` [p.271]

11.8.2.16 public synchronized void open(boolean trackAllServices)

trackAllServices If true, then this ServiceTracker will track all matching services regardless of class loader accessibility. If false, then this ServiceTracker will only track matching services which are class loader accessibile to the bundle whose BundleContext is used by this ServiceTracker.

- 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.

Since 1.3

11.8.2.17 public void remove(ServiceReference reference)

reference Reference to the service to be removed.

- Remove a service from this ServiceTracker object. The specified service will be removed from this ServiceTracker object. If the specified service was being tracked then the ServiceTrackerCustomizer.removedService method will be called for that service.

11.8.2.18 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.263]

11.8.2.19 public int size( )

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

Returns Number of services being tracked.

11.8.2.20 public Object waitForService(long timeout) throws InterruptedException

timeout time interval in milliseconds to wait. If zero, the method will wait indefinately.

- Wait for at least one service to be tracked by this ServiceTracker object.

It is strongly recommended that waitForService is not used during the calling of the BundleActivator methods. BundleActivator methods are expected to complete in a short period of time.

Returns Returns the result of getService().

Throws `IllegalArgumentException` – If the value of timeout is negative.

11.8.3 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.

11.8.3.1 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.

11.8.3.2 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.

11.8.3.3 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