OSGi Service Gateway Specification

Release 1.0

May, 2000

Contents

OSGi Service Gateway 1.0: Framework Specification

Contents

Chapter 1 *Overview*

The primary goal of the OSGi service framework ("Framework") is to use the Java (TM) programming language's platform independence and dynamic code-loading capability to make development and dynamic deployment of applications for small-memory devices easier.

This goal is met in two ways. First, the Framework provides a consistent programming model during application development; it supports the development and use of services by decoupling a service's specification (the interface) from its implementation. Thereby:

- Developers can provide multiple implementations to the same service interface
- Developers who use a service can code against that service's interface without regard to its implementation

This support is critical, because the Framework is designed to run on a variety of devices; the different hardware characteristics of the devices could affect many aspects of the service implementation, yet a stable service interface ensures the stability of the overall software system running on the device.

For example, on a high-end device, a logging service might be able to store log messages on a hard drive, while on a disk-less device, the log entries may have to be saved remotely. The developers of the two logging service implementations implement the same interface; the developers of services that use a logging service write code against the logging service interface, without regard to which implementation their service might use.

Second, it provides life-cycle management functionality that permits application developers to partition applications into small self-installable components. These components are called bundles. Bundles can be downloaded on demand and removed when they are no longer needed. When a bundle is installed and activated in the Framework, it can register any number of services that can be used by other bundles.

This dynamic aspect makes the software extensible on the device after deployment: new bundles can be installed for added features or existing bundles can be updated for bugfixes without bringing down the entire system.

Chapter 2 *Concepts*

The Framework is designed for creating extensible services using the Java programming language. To take best advantage, developers should design an application as a set of bundles that contain services, with each service implementing a segment of the overall functionality. These bundles are then downloaded on demand by the target device. For example, a text editing application designed this way may rely on a spell-checking service. The editor would instruct the Framework to download a spell-checker for it to use.

The key entities in the Framework are:

- Services - The Java classes that perform certain functionality, usually written with interface and its implementation separated.
- Bundles - The functional and deployment unit for shipping services
- Bundle contexts - The execution environments of the bundles in the Framework

2.1 Services

A service is a self contained component, accessible via a defined service interface. In the OSGi model, an application is built around a set of cooperating services: it can extend its functionality at runtime by requesting more services which it requires. The Framework maintains a set of mappings from services to their implementations and has a simple query mechanism (LDAP based syntax) that enables an installed service to request and use the available services. The Framework manages the dependencies among services.

A developer defines a service as an interface and provides its implementation. Then she can register the service with the Framework. (Registering a service is also called publishing it) When a service is registered, it can be given a set of properties (name/value pairs) to enable a sophisticated retrieval based on LDAP attribute comparisons.

After a service is published, other services can use it to accomplish their tasks; they look up the service from the Framework with a search filter, and will get back the matching service references. The service reference can then be used to get a Java object that implements the desired service.

Note that the Framework does not actually give out references to objects implementing a service directly, which would also instantly create a dynamic dependency on the bundle providing the service. Instead, it gives out a `ServiceReference` object, which can be stored and passed on to other bundles, without the implications of dependencies. When the service is actually to be used by a bundle, a reference to the implementing object can be obtained from the current `BundleContext`, passing it the `ServiceReference`.

2.2 Bundles

To be available to the Framework, a service implementation must be packaged. Service implementations are packaged into *bundles*. A bundle is a JAR file that:

- Contains the resources implementing zero or more services. These resources may be class files for the Java programming language, as well as any other data (such as HTML help files, icons, and so on).
- Contains a Manifest file with headers specifying various parameters so that the Framework can correctly install and activate the bundle. These parameters include what Java packages this bundle provides, what packages it needs, how to locate its bundle activator, and so on.

For each bundle installed in the Framework, there is an associated `Bundle` object. This object is used to manage the namespace of the bundle's Java classes, by directing the loading and resolution of those classes. By establishing separate namespaces for bundles, class-name conflicts among bundles are avoided.

Using the naming conventions described in section 6.8 of the Java Language Specification also avoids conflicts. The Framework provides its scoping as an additional precaution: if two bundles have class names in common, the fact that those names are scoped by different bundle namespaces means that there is no contention.

A bundle is also used to get information about the current bundle lifecycle status and to start, stop and update bundles.

2.2.1 Bundle Context

The relationship between the Framework and its installed bundles is captured by the notion of a bundle context.

A bundle context (or simply context) is the execution environment of a bundle within the Framework. It is created by the Framework when the bundle is activated and serves the following purposes:

- **To install new bundles into the Framework.**
- **To register services in the Framework's registry.**
- **To retrieve references to registered services.**
- **To subscribe or unsubscribe to the events broadcast by the Framework.**
- **To find out about other bundles installed in the Framework.**
- **To supply persistent storage to the installed bundle.** A private persistent storage area is associated with each bundle context, which bundle programmers can access. This storage area can be used to store and retrieve data (as files) across invocations of the underlying Java Virtual Machine.

Chapter 3 *Bundles*

A bundle contains services, Java classes, and other resources; it is deployed as a JAR file and represents a functional component when it is installed and activated inside the Framework.

The Framework may be configured with a bundle providing administrative capabilities, which may be used to manage the lifecycle of other bundles.

Bundles can share Java classes. They make this arrangement by declaring appropriate import/export headers in their Manifest files. A bundle is *resolved* when all the classes and/or native code it needs are made available to it. It cannot be started until resolved.

The Framework creates a `Bundle` object for each installed bundle, and a `BundleContext` object for a bundle when it gets activated. `Bundle` is the interface of an installed bundle to its users, i.e., other bundles or the Framework itself. The `BundleContext` interface, on the other hand, represents the execution environment of a bundle within the Framework, and acts as a proxy to the underlying Framework.

A bundle programmer can customize her bundle's start/stop logic by implementing the `BundleActivator` interface.

Once a bundle has become active, its functionality manifests itself, and the services it provides are made available to other bundles in the Framework.

All these aspects of a bundle are discussed in the following sections.

3.1 The State Transitions of a Bundle

A bundle may be in one of six states:

- INSTALLED - This is the state after a bundle has been successfully installed.
- RESOLVED - This is the state when all the Java classes and/or native code that this bundle requires have been made available to it. Usually this state means that a bundle is ready to be started; a stopped bundle will be put back into this state too.
- STARTING - This indicates that the bundle is being started.
- STOPPING - This indicates that the bundle is being stopped.
- ACTIVE - A bundle has successfully started and is running.
- UNINSTALLED - The bundle has been uninstalled. It cannot move into any other states.

Whether a bundle has been started or stopped is persistently recorded: a bundle in the activated state remains in that state across restarts of the Framework, until someone stops or uninstalls that bundle.

3.2 Installing a Bundle

The `installBundle` method of the `BundleContext` interface installs a new bundle.

Installing a bundle within the Framework must be a persistent operation: the bundle must remain installed across Java Virtual Machine invocations, until it is explicitly uninstalled. The installation process must be atomic: either this method completely installs the bundle or, if the installation fails, the Framework must be left in the state before this install method is called.

Once a bundle is installed, an instance of `Bundle` is created; the rest of the lifecycle operations are performed against that `Bundle` object.

The Framework implementation of the `installBundle` method must:

1. Check that the bundle is not already installed.

   If a bundle with the same location string has already been installed into the Framework, return that bundle.

   The location string is used to uniquely identify an installed bundle. For example, if the location string is the

URL from which a bundle is obtained, a copy of the bundle loaded from a mirror site with a different URL will be installed a second time. Determining whether this is the proper behavior is the responsibility of the management software at deployment time; it is outside the scope of this specification.

2. Obtain the bundle's contents.

   If an `InputStream` is specified, the Framework will try to fetch the bundle from the supplied stream. Otherwise, the Framework will try to download the bundle using its location string, which typically is the string representation of a URL identifying the bundle's codebase. If the bundle cannot be obtained, go to Step 7.
3. Create a `Bundle` for the bundle and allocate its associated resources.

   The associated resources consist of at least a unique identifier. If file system support is available on the platform, a bundle-private storage area should be allocated. If this step fails, continue with Step 7.
4. Set the bundle's state to INSTALLED.
5. Resolve the bundle's classpath and native code dependencies, if any.

   The Framework makes available to the bundle its classpath JAR files and native code libraries that the bundle has specified in its `Bundle-ClassPath` and `Bundle-NativeCode` Manifest headers, respectively (see section Resolving a Bundle for details). The specified JAR files and native code libraries must be contained in the bundle's own JAR file.

   If this step fails, continue with Step 7.

   If the bundle does not require any classes from other bundles, set its state to RESOLVED.

   If the bundle does require classes from other bundles, the Framework may attempt to resolve these classes at this time, or it may wait until the bundle gets started.
6. Broadcast a `BundleEvent` of type `BundleEvent.INSTALLED` by calling the `bundleChanged` method of each registered `BundleListener`.
7. Finish the execution of this method, ensuring atomicity:
   - If the installation was free of errors, return the bundle created in Step 3.
   - If there has been an error, undo any work performed by this method and throw a `BundleException`.

3.3 Resolving a Bundle

The Framework *resolves* a bundle when the classes and services it needs are made available to it.

The Framework allows a bundle to declare the following types of dependencies:

1. **Classpath dependencies**: The bundle uses a set of libraries for the Java platform. These libraries, which are packaged as JAR files, must be contained as resources in the bundle's own JAR file.
2. **Native code dependencies**: The bundle uses native code libraries. These libraries must be contained as resources in the bundle's own JAR file.

   The above dependencies can be resolved when the bundle is first installed.
3. **Package dependencies**: A bundle requires classes from other bundles. It cannot be started unless a set of packages (optionally conforming to a given specification version) is being exported within the Framework.

Let's take a look at how to specify the dependency information in a bundle, and how the Framework resolves those dependencies.

Classpath Dependencies

The `Bundle-ClassPath` header allows a bundle to extend its classpath with one or more JAR files that it contains. It is a list of comma-separated path names. A path name can be either dot ('.') (which represents the bundle's own JAR file), or it can be the path of a JAR file contained in the bundle's JAR file. The JAR files listed will be searched in the order specified. If `Bundle-ClassPath` is not specified, the default value is dot. If `Bundle-ClassPath` is specified, but dot is not an element of it, the bundle's main JAR file will not be searched; instead, only the JAR files referenced by the path name elements (and contained within the bundle's main JAR file) will be searched.

The value of the header must conform to the following syntax:

```
    Bundle-ClassPath: path ( , path )*
    path: string <"/"-separated path name identifying a JAR file>
```

For example, the following declaration:

```
    Bundle-ClassPath: com/sun/jes/impl/http/servlet.jar
```

would make all the classes defined in `servlet.jar` available to the bundle.

Native Code Dependencies

The `Bundle-NativeCode` header allows a bundle to carry the native code it needs, and make use of it when it is installed. The bundle must have `RuntimePermission` in order to run native code in the Framework. The value of the header must conform to the following syntax:

```
    Bundle-NativeCode: nativecode-clause ( ,nativecode-clause )*
    nativecode-clause: nativepaths ( ; env-parameter )*
    nativepaths: nativepath ( ; nativepath )*
    env-parameter: ( processordef | osnamedef | osversiondef |
    languagedef )
    processordef: processor=token
    osnamedef: osname=token
    osversiondef: osversion=token
    languagedef: language=token
```

For example:

```
    Bundle-NativeCode: http.DLL ; osname=Win95; processor=x86; language=en,
                       http.so; osname=Solaris; processor=sparc
```

The `Bundle-NativeCode` header allows a bundle programmer to specify an environment, and to declare what native code libraries it carries for that specific environment. The environment is characterized by the following properties:

- processor: The processor on which the hosting the Framework is running
- osname: The operating system
- osversion: The version of the operating system
- language: The language

The Framework uses the following algorithm to find the "best" matching native code clause:

1. Pick the clauses with a matching processor and operating system with the one the Framework runs on.

   The specified processor name and operating system name are matched against the system properties `org.osgi.framework.processor` and `org.osgi.framework.os.name`. If no clause matches both the required processor and operating system, the bundle installation/activation fails. If only one clause matches, it can be used, otherwise, remaining steps are executed.
2. Pick the clauses that best match the operating system version.

   The specified operating system version will be compared to the value specified in the `org.osgi.framework.os.version` property. If they match each other exactly, that clause is considered the best match. If there is only one clause with an exact match, it can be used. If there is more than one clause that matches the property, these clauses will be picked to perform the next step. Operating system versions are taken to be backward compatible. If there is no exact match in the clauses, clauses with operating system versions lower than the value specified in `org.osgi.framework.os.version` will be picked. If there is only one clause which has a compatible operating system version, it can be used. Otherwise, all clauses with compatible operating system versions will go through the next step. If no clause has a matching or compatible operating system version, pick the clause that does not have operating system version specified. If that is not possible, the bundle installation fails.
3. Pick the clause that best matches the language.

   The specified language is matched against the system property `org.osgi.framework.language`. If more than one clause remains at that point, then the Framework is free to pick amongst them randomly. If no clauses have the exact match with the value of the property, pick the clause that does not have language specified. If that is not possible, the bundle installation fails.

Package Dependencies

A bundle declares what it needs using `Import-Package` headers, and declares what it provides to other bundles using `Export-Package` headers in its Manifest file. In its `Import-Package` Manifest header, a bundle must specify all packages that it requires whose names do not start with `java.` .

Exporting Packages

A bundle can make available to other bundles a set of Java packages. When a package is exported by a bundle, the Framework guarantees that its classes will always be loaded from the exporting bundle, and that they will be shared across the bundles that import the package (see below). If there is more than one bundle requesting to export the package, the Framework can decide which bundle to select as the exporter of the package. The Framework should pick the bundle with the highest version of that package.

In order to export a package, a bundle programmer must specify an `Export-Package` header conforming to the following syntax:

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

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

                    package-name: string <fully qualified Java package name>

                    parameter: token=string
```

The only parameter `token` recognized by this version of the Framework is the string `specification-version`. Its `string` value must conform to the semantics described in the Java 2 Package Versioning Specification.

As an example, this `Export-Package` header:

```
    Export-Package: javax.servlet; specification-version="2.0",
                    javax.servlet.http; specification-version="2.0",
                    org.osgi.service.http
```

specifies that the bundle provides all the classes defined by version 2.0 of the `javax.servlet` and `javax.servlet.http` packages, as well as any version of the classes defined in the `org.osgi.service.http` package.

In order to be able to export a package, a bundle must have the `PackagePermission` to `export`.

A bundle that exports a package implicitly imports the same package name and version level that it exports. Exported packages from a bundle are available to other bundles as soon as that bundle completes installation on the Framework.

Although a bundle can export all its classes to other bundles, this practice is discouraged except in the case of very stable library packages that will not need updating. The reason is that the Framework may not be able to promptly reclaim the space occupied by the exported classes if the bundle is uninstalled. Designs that separate interfaces and their implementations are preferred. One should put the interface into a separate Java package to be exported, while keeping the implementation classes hidden from the outside world. If the same interface has multiple implementations in multiple bundles, the bundle programmer can package the interface into all of these bundles -- the Framework will load the interface class from one and only one of the bundles. Interfaces that have the same class name should have exactly the same method signatures. Since a modification to an interface affects all its callers, interfaces should be designed carefully and should remain stable.

Importing Packages

The `Import-Package` header can be used by bundles in order to request access to a given set of Java packages. Bundles can import packages that have been exported by other installed bundles. The Framework guarantees that at any time a bundle is only exposed to one version of a package it has imported.

The syntax of the `Import-Package` header is as follows:

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

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

                    package-name: string <fully qualified Java package name>

                    parameter: token=string
```

The only parameter `token` recognized by this version of the Framework is the string `specification-version`. Its `string` value must conform to the semantics described in the Java 2 Package Versioning Specification.

As an example, this `Import-Package` header:

```
Import-Package: javax.servlet ; specification-version="2.0"
```

requires that the bundle be resolved against all the classes defined in version 2.0 or above of the `javax.servlet` package.

A bundle will implicitly import the same package name and version level as it exports. If the bundle can use a lower specification version of the package than it exports, then an `Import-Package` header should be specified containing the package name and the lower specification version. Otherwise, a separate `Import-Package` header for this package is unnecessary.

In order to be able to import a package, a bundle must have the `PackagePermission` to `import`.

Exporting and Importing Services

The following Manifest headers can be defined:

- `Export-Service: class-name (, class-name )*`

  If present, this header describes the services the bundle may register. This header provides advisory information that is not used by the Framework. It is intended for use by server-side management tools.
- `Import-Service: class-name (, class-name )*`

  If present, this header describes the services the bundle may use. This header provides advisory information that is not used by the Framework. It is intended for use by server-side management tools.

3.4 Starting and Stopping a Bundle

Starting a Bundle

The Framework's implementation of the `Bundle.start` method must:

1. Check the bundle's state using the bundle's `getState` method:
   - If the bundle is UNINSTALLED, throw an `IllegalStateException`.
   - If the bundle is STARTING or STOPPING, wait for it to change its state before continuing. If this does not happen within a reasonable time, throw a `BundleException`.
   - If the bundle is ACTIVE, return.
2. If the bundle is INSTALLED, resolve its dependencies. If that fails, throw a `BundleException`.
3. Set the state of the bundle to STARTING.
4. Invoke the `BundleActivator`'s `start` method.

   This step is skipped if the bundle doesn't define an activator. If the `BundleActivator`'s `start` method fails, throw a `BundleException`.

   Do not continue with the remaining steps if a `BundleException` has been thrown. The bundle cannot be started.
5. Record persistently that this bundle has been started.

   When the Framework is restarted, the bundle is started automatically.
6. Set the bundle's state to ACTIVE.
7. Broadcast a `BundleEvent` of type `BundleEvent.STARTED` by calling the `bundleChanged` method of each registered `BundleListener`.

This method must be atomic. The caller must have `AdminPermission` to start a bundle.

Stopping a Bundle

The Framework's implementation of the `Bundle.stop` method must:

1. Check the bundle's state:
   - If the bundle is UNINSTALLED, throw an `IllegalStateException`.
   - If the bundle is STARTING or STOPPING, wait for it to change its state before continuing. If this does not happen within a reasonable time, throw a `BundleException`.
   - If the bundle is not ACTIVE, return.
2. Set the state of the bundle to STOPPING.
3. Record persistently that this bundle is stopped, so that it will not be started automatically when the Framework restarts.
4. If the bundle defines an `BundleActivator`, run its `stop` method.

   Catch any exceptions that the `BundleActivator`'s `stop` method may throw, so that this method can complete the remaining steps.
5. Unregister any event listeners and remaining services registered by this bundle; release any services used by this bundle.
6. Set the state of the bundle to RESOLVED.
7. Broadcast a `BundleEvent` of type `BundleEvent.STOPPED` by calling the `bundleChanged` method of each registered `BundleListener`.
8. If the `BundleActivator`'s `stop` method has thrown an exception, throw a `BundleException`.

The `stop` method must be atomic.

`BundleActivator` Interface

The `BundleActivator` interface describes methods that the Framework runs when it starts and stops a bundle. An activator of a bundle will only be run if it is resolved, i.e., all the required dependencies of the bundle (as stated in its Manifest) are available.

Here is the full definition of the `BundleActivator` interface.

To inform the Framework of the fully qualified class name serving as its bundle activator, a bundle programmer must specify a `Bundle-Activator` header in the bundle's Manifest. For example:

```
Bundle-Activator: com.myCompany.bundle1.myBundleActivator
```

The `BundleActivator` interface allows bundle programmers to perform specific tasks when a bundle is started or stopped. Supplying a class that implements this interface is optional. A library bundle usually does not need to define an activator, but a service-providing bundle must because this is the only way for it to access its bundle context, which is passed in from `BundleActivator.start` and `BundleActivator.stop` methods.

The Framework guarantees that if a `BundleActivator` instance's `start` method has executed successfully, that same instance will receive a call to its `stop` method when the bundle is deactivated.

The `BundleActivator`'s `start` method can allocate any resources that the bundle requires, and usually registers the services provided by the bundle. If its `start` method does not register any services, the bundle can register its services at a later time, while it is ACTIVE.

In general, the `stop` method of the `BundleActivator` interface is supposed to undo all the actions of the `start` method: it should release any resources allocated during activation. Note that the `BundleActivator`'s `stop` method is permitted to unregister services, but there is no requirement for it to do so. If the bundle registered any services, either through its `BundleActivator`'s `start` method or while the bundle was ACTIVE, the Framework will unregister them when the bundle is stopped.

A class that implements the `BundleActivator` interface must be public and must have a public default constructor.

3.5 Updating a Bundle

The Framework must support updating an installed bundle. The update process is meant to support migration from one version of a bundle to a newer, backward compatible version of the same bundle. A bundle, `bundle1`, is backward compatible with another bundle, `bundle2`, if `bundle1` provides at least the services provided by `bundle2`, and each service interface in `bundle1` is compatible (as defined in section 13.5 of the Java Language Specification) with its counterpart in `bundle2`. If `bundle1` exports any packages, then `bundle2` must also export at least the same set of packages, and each of these packages must be compatible with their respective counterparts in `bundle1`.

A Framework implementation must guarantee that only one version of a bundle's classes are available at a time.

The `update` method of the `Bundle` interface performs the update operation on a bundle.

The Framework implementation of the `update` method must:

1. Check the bundle's state:
   - If the bundle is ACTIVE, stop it as described in the `Bundle.stop` method; if this throws an exception, rethrow it and return.
   - If the bundle is UNINSTALLED, throw an `IllegalStateException`.
2. Back up the bundle's JAR file. If the backup fails, continue with Step 6.2.
3. Install the new version of the bundle.

   If an `InputStream` is supplied, get the bundle from the stream. Otherwise, if the bundle has a `Bundle-UpdateLocation` Manifest header, use its value to retrieve the new bundle. Otherwise, call the bundle's `getLocation` method to determine its original location and retrieve the new version from there.
4. Set the state of the bundle to INSTALLED
5. If the update process has succeeded:
   1. Broadcast a `BundleEvent` of type `BundleEvent.UPDATED` by calling the `bundleChanged` method of each registered `BundleListener`.
   2. Resume the target bundle if it is initially active. This step performs the equivalent of the `Bundle.start` method.
6. If the update process has failed:
   1. Revert the bundle to its old version.
   2. Resume the given bundle if it is initially active. This step performs the equivalent of the `Bundle.start` method.
   3. Throw a `BundleException` to indicate that the update has failed.

3.6 Uninstalling a Bundle

The Framework implementation of the `uninstall` method of the `Bundle` interface must:

1. Stop this bundle if it is ACTIVE.

   After completing the remaining steps, throw a `BundleException` if stopping this bundle has failed.
2. Broadcast a `BundleEvent` of type `BundleEvent.UNINSTALLED` by calling the `bundleChanged` method of each registered `BundleListener`.
3. Set the state of the bundle to UNINSTALLED.
4. Release any persistent resources the bundle was holding.

Note that this method must always succeed. Once this method returns, the Framework state must be the same as if the bundle had never been installed.

3.7 When the Framework Starts and Stops

When the Framework is started, the following actions occur:

1. Event handling is enabled. Events can now be delivered to listeners.
2. The Framework persistently records whether an installed bundle has been started or stopped. When the Framework is restarted, all installed bundles previously recorded as being ACTIVE will be started automatically as described in the `Bundle.start` method. Reports any exceptions that occur during startup using events of type `FrameworkEvent`.
3. A `FrameworkEvent` of type `STARTED` is broadcast.

When the Framework is stopped, the following actions occur:

1. Suspend all ACTIVE bundles as described in the `Bundle.stop` method, except that their persistently recorded state remains ACTIVE. Those bundles will be restarted when the Framework is next started. Reports any exceptions that occur during stopping using events of type `FrameworkEvent`.
2. Event handling is disabled.

3.8 Using a Bundle

The Framework creates a `Bundle` object for each installed bundle. `Bundle` is the interface of an installed bundle to its external users (such as the Framework or other bundles). Note that bundle access is not restricted: any bundle can enumerate the set of installed bundles; however, information that allows to identify a bundle (such as its location, or its header information) is only provided to bundles that have the `AdminPermission`.

The complete definition of the `Bundle` interface is here. So far we have covered the `start`, `stop`, `update`, and `uninstall` methods of the `Bundle` interface. Now we will explain the remaining methods of this interface.

Accessing the Attributes and Resources of a Bundle

Bundles have the following attributes:

- A bundle identifier, which must be unique and persistent and must have the following properties:
  - The identifier is a `long`. The `long` zero (0) is reserved for use by the Framework.

    - Once its value is assigned to a bundle, that value must not be reused for another bundle, even if the original bundle is uninstalled.
    - Its value must not change as long as the bundle remains installed.
    - Its value must not change when the bundle is updated.
- A location identifier: the location string of the bundle. This could be the string representation of the location where the bundle was obtained.
- The state of the bundle.
- Human readable information about the bundle.

A bundle exposes the following resources:

- Services that the bundle depends on.
- Services that the bundle has registered with the Framework.
- Any resources contained in the bundle's JAR file that are part of an exported package (as declared in the `Export-Package` header of the bundle's Manifest).

Getting a Bundle's Identifier

The `getBundleId` method must return the unique identifier that the Framework assigned to this bundle when the Framework installs it.

Getting a Bundle's Location Identifier

The `getLocation` method returns the location identifier of the current version of the bundle. This is typically a string representation of the URL where this bundle is available. Note that unlike the bundle's unique identifier, the location of a bundle may change when the bundle is updated. Calling this method while the Framework is updating the bundle results in undefined behavior.

Getting a Bundle's State

The `getState` method returns the state of the bundle. Use the following code to determine which state a bundle is in:

```
    if ((b.getState() & RESOLVED) != 0)
```

The state is expressed as a bit string so that one can determine if the bundle is in a set of states conveniently; at any time a bundle can only be in one state physically.

Getting a Bundle's Manifest Headers

In addition to predefined Manifest headers (such as `Export-Package` or `Bundle-Activator`), the Framework allows bundle programmers to supply information about their bundles; the following are the Manifest headers understood by the Framework:

- **Bundle-Name**: A short, human readable name for the bundle,
- **Bundle-Description**: A short description of what the bundle does,
- **Bundle-Vendor**: The name of the bundle's vendor,
- **Bundle-Version**: The version of the bundle.

  This version is a soft version number that serves as a hint. The Framework does not use this version number to determine if a bundle should be loaded.

  The soft version number should consist of a dot (.) separated list of numbers, encoded as a string. It should follow the rules defined in the Java 2 description of the `java.lang.package` class:

Major version numbers - each increment signifies significant functional changes, which are not necessarily compatible with previous releases.

Minor version numbers - each increment signifies smaller extensions to the functionality, that are compatible with previous releases of the same major version.

Micro versions - each increment signifies bug fixes that do not change the functionality of the package (except by making available any intended functionality that was inaccessible due to a bug).

The list of numbers should be provided from most significant (major) to least significant (micro).

- **Bundle-DocURL**: A URL that provides some documentation about the bundle,
- **Bundle-ContactAddress**: A contact address for reporting problems with the bundle.

Any additional headers can be defined by the bundle programmer as needed.

Every header is optional. Missing headers correspond to the `null` value. The `getHeaders` method returns a `Dictionary` instance that contains name-value pairs. This method requires the `AdminPermission`, because some of the header information (e.g., the packages listed in the `Export-Package` header) may be sensitive.

The `getBundleId`, `getLocation`, and `getHeaders` methods will continue to provide the respective information when the bundle is in UNINSTALLED state.

Getting Information about Services associated with a Bundle

The `getRegisteredServices` method returns the services that this bundle has registered with the Framework.

The `getServicesInUse` method returns the services that this bundle is using.

Checking a Bundle's Permissions

The `hasPermission` method returns `true` if the bundle has the specified permission, and `false` if it does not have that permission, or the parameter 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 permissions.

3.9 Using a Bundle Context

The `BundleContext` interface represents the execution environment of a bundle within the Framework. It is created when a bundle is activated and passed to the bundle in the `BundleActivator.start` and `BundleActivator.stop` methods. A `BundleContext` object is made invalid and should not be used after its associated bundle has been stopped.

The complete definition of the `BundleContext` interface can be found here.

We have already covered the `installBundle` method of `BundleContext`. The `BundleContext` interface also provides methods to subscribe to events broadcast from the Framework, register services with the Framework, and retrieve service references from the Framework's service registry. Those methods are covered in the sections on events and services, respectively. The remaining methods of `BundleContext` are discussed here:

Accessing Information about Bundles in the Framework

- The `getBundle` method returns the `Bundle` object associated with the `BundleContext`.
- The `getBundles` method returns an array of the bundles currently installed. It returns `null` if there are no bundles installed.
- The `getBundle(int id)` method returns the `Bundle` instance with the specified unique identifier, or `null` if no matching bundle is found.

Accessing a Bundle's Persistent Storage

On platforms with file system support, the Framework should provide a persistent storage area for each installed bundle. The `BundleContext` interface defines access to this storage in terms of the `java.io.File` class of the Java programming language. This does not mean that the device hosting the Framework must have a disk, but rather that it must have some form of persistent storage that can be accessed as a file system.

The `getDataFile` method takes a relative file name as an argument and must translate it into an absolute file name under the root of the bundle context's persistence storage area and return the path that it finds.

This method will return `null` if there is no underlying file system for this platform.

Retrieving Environment Properties

The Framework defines the following environment properties which can be queried using the `getProperty` method:

- `org.osgi.framework.version`: The version of the Framework.
- `org.osgi.framework.vendor`: The vendor of this Framework implementation.
- `org.osgi.framework.language`: The language being used. See ISO 639 for possible values.
- `org.osgi.framework.os.name`: The name of the operating system of the hosting computer.
- `org.osgi.framework.os.version`: The version number of the operating system of the hosting computer.
- `org.osgi.framework.processor`: The name of the processor of the hosting computer.

The last four properties are used by the `Bundle-NativeCode` manifest header's matching algorithm for selecting native code to determine if a bundle that carries native code can be run in a given Framework environment.

3.10 Summary

The following table describes which Interface methods and Manifest headers apply to the various stages in the lifecycle of a bundle:

Bundle Stage	Methods that Programmer May Define to Perform Bundle-Specific Tasks	Manifest Headers that Provide Bundle-Specific Data
Using bundle's own resources		`Bundle-ClassPath` `Bundle-NativeCode`
Using resources from other bundles		`Import-Package`
Providing resources to other bundles		`Export-Package`

Using/providing services *(advisory)*		`Import-Service` `Export-Service`
STARTING	`BundleActivator.start`	`Bundle-Activator`

INSTALLED, RESOLVED, STARTING, ACTIVE, STOPPING, UNINSTALLED		`Bundle-Name`
		`Bundle-Description`
		`Bundle-Version`
		`Bundle-Vendor`
		`Bundle-DocURL`
		`Bundle-ContactAddress`

Bundle update		`Bundle-UpdateLocation`
STOPPING	`BundleActivator.stop`	`Bundle-Activator`

Chapter 4 *Services*

A service is an object that provides some well-defined functionality which is defined by the object classes it implements. It is owned by (and runs within) a bundle that makes its functionality available to other bundles by registering it with the Framework. The dependencies between the bundle owning the service and the bundles using it are managed by the Framework.

4.1 Registering a Service

A bundle makes a service available to other bundles by *registering* it with the Framework. Registering a service is also referred to as *publishing* it. A service is registered using the names of the Java object types (the names of Java interfaces or classes) that the service object is an instance of and can be cast to.

The Framework allows its bundles to dynamically register and unregister services. A bundle can register a service while the Framework is activating the bundle (see Starting a Bundle), or at any time during its active phase. An active bundle can also remove one or more of its services from the Framework's service registry at any time and for any reason. All of a bundle's services are removed from the service registry when the bundle is stopped (see Stopping a Bundle).

A bundle registers a service with the Framework by calling its `BundleContext`'s `registerService` method. A service object can be registered under a single class or multiple classes of which it is an instance. The names of the classes under which a bundle wants to register its service are provided as arguments to the `registerService` method. The Framework ensures that the service object being registered actually is an instance of all the classes specified.

The service being registered may be further refined by a dictionary object (i.e., an object of type `java.util.Dictionary`), which describes the properties of the service as a collection of *key/value* pairs.

Service registration is subject to a security check: The registering bundle must have the `ServicePermission` (with action `register`) to register the service under all the class names specified. Otherwise, the service will not be registerd, and a `SecurityException` will be raised.

The class(es) under which a service has been registered successfully are automatically added (by the Framework) to its dictionary under the keyword `objectClass`.

If the service registration succeeds, the Framework returns a `ServiceRegistration` object to the registering bundle. A service can be unregistered only by the entity (bundle) that holds its `ServiceRegistration`.

Every successful service registration yields a unique `ServiceRegistration` object, even when the *same* service object is registered multiple times.

The `ServiceRegistration` object may also be used to update the properties of a service after it has been registered. Actually, this is the only reliable way to change the properties of a service after it has been registered: Modifying a service's dictionary after the service was registered may not have any effect on the registered service.

Note that if a `ServiceRegistration` is garbage-collected, the Framework may remove the service from its registry. Therefore, if a bundle wants to keep its service registered, it must keep the `ServiceRegistration` object referenced.

A `ServiceReference` to the registered service can be obtained by calling the `getReference` method on the `ServiceRegistration`. From the `ServiceReference`, the actual service object can be retrieved as described in the Obtaining a Service section.

4.2 Obtaining a Service

In order to use a service object and call its methods, a bundle must first obtain a `ServiceReference` object of the service.

A `ServiceReference` object can be obtained either from a `ServiceRegistration` (see section Registering a Service) or from the service registry managed by the Framework.

In order to obtain a `ServiceReference` from the Framework, a bundle calls the `getServiceReference` or `getServiceReferences` methods on its `BundleContext`, which are defined as follows:

- `getServiceReference` returns a `ServiceReference` to a service that implements the class specified as a parameter.
- `getServiceReferences` returns an array of `ServiceReference` objects to services that implement the class and satisfy the search filter specified as parameters. The filter is the string representation of an LDAP search filter as defined in RFC 1960: A String Representation of LDAP Search Filters.

Both methods require that the caller have the `ServicePermission` to `get` the service for the specified class name. If the caller lacks the required permission, `null` is returned. (Note that there is no permission check on getting a `ServiceReference` from a `ServiceRegistration`.)

A `ServiceReference` is valid only as long as the service it references has not been unregistered.

The `ServiceReference` object can be used to find out more information about the service it references by calling its `getPropertyKeys` and `getProperty` methods: The former returns a list of all the service's property keys, which can be passed to the latter to determine the corresponding property value. Both methods must continue to provide information about the referenced service even after the service has been unregistered from the Framework. This may be useful in the case where a `ServiceReference` is stored with the log service.

Once a bundle has obtained a `ServiceReference` object, it can retrieve the corresponding service object by calling the `getService` method on its `BundleContext`. This method:

- returns `null` if the underlying service has already been unregistered;
- checks that the caller has the `ServicePermission` to `get` the service under at least one of the classes under which it was registered (this information is available from the dictionary object associated with the

service); this permission check is necessary so that `ServiceReference` objects can be passed around freely, without compromising security;
- increments the bundle's usage count of the service by one.

After this method has returned, the bundle owning the `BundleContext` on which the method was called has established a *dynamic service dependency* on the service and on the bundle that registered it. The dependent bundle can remove its service dependency by releasing its use of the service. Before a service is removed from the service registry, bundles with a dynamic dependency on it have the opportunity to release the service in order to remove their dependency on it.

A service can be used by casting it to one of the Java object types that it is an instance of and calling the methods defined by that type.

4.3 Releasing a Service

In order to remove its dynamic dependency on a service, a bundle must release the service by calling `BundleContext.ungetService`, passing it the service's `ServiceReference` object.

`BundleContext.ungetService` returns a `boolean` value: If the bundle's usage count of the service already is zero when the method is being called, or the service has already been unregistered, `false` is returned. Otherwise, the bundle's usage count of the service is decremented by one, and `true` is returned. If the bundle's usage count of the service has dropped to zero, and the service had been registered as a `ServiceFactory`, the factory's `ungetService` method is called to release the service object for the calling bundle.

4.4 Service Factories

Service factories allow bundles to customize the services they provide to other bundles by returning a service instance that is specific to the requesting bundle. To take advantage of this concept, a bundle developer has her service object implement the `ServiceFactory` interface.

The `ServiceFactory` interface defines two methods: `getService` and `ungetService`, which can be accessed by the service registry only:

- If a call to `BundleContext.getService` is made, where the given `ServiceReference` points to a service that implements the `ServiceFactory` interface and the bundle's usage count of that service is zero (i.e., the bundle currently does not have any dynamic dependencies on the service), that call is routed (by the Framework) to the `getService` method of the `ServiceFactory`, passing it the calling bundle as a parameter. The `ServiceFactory` has a choice of returning a service object that is specific to the calling bundle or one that is the same for all bundles. The Framework will cache the mapping of requesting bundle to service object, and return the cached service object to the bundle on future calls to `BundleContext.getService` with the same (or a corresponding) `ServiceReference` (as long as that bundle's usage count of the service is greater than zero).
- When a call to `BundleContext.ungetService` is made, where the given `ServiceReference` points to a service that implements the `ServiceFactory` interface and the bundle's usage count of that service will drop to zero after this call returns (i.e., the bundle is about to release its last dynamic dependency on the service), that call is routed (by the Framework) to the `ungetService` method of the `ServiceFactory`, allowing the `ServiceFactory` to release the service object it had created for the calling bundle. Also, the cached copy of the service object in the Framework will be removed.

Service factories help manage dependencies that are not explicitly managed by the Framework. By having a returned service object be bound to the requesting bundle, the service can listen to events related to that bundle and remove objects (e.g., listeners) registered with it (by that bundle) when the bundle goes away.

4.5 Unregistering a Service

A service can be unregistered by calling the `unregister` method on its `ServiceRegistration`. This is to ensure that only the bundle that is holding the `ServiceRegistration` can unregister the associated service. This also means that the bundle that unregisters a service does not have to correspond to the bundle that registered it: The registering bundle could have passed the `ServiceRegistration` object to another bundle for it to unregister the service.

The service that is being unregistered will be removed from the Framework's service registry. As a consequence, all `ServiceReference` objects obtained for that service may no longer be used to get the service object; instead, their `getService` method will return `null`.

The service will be unregistered regardless of whether or not other bundles currently depend on it. However, bundles that depend on the service will be notified of its imminent unregistration by a service event of type `ServiceEvent.UNREGISTERING`, which is sent synchronously, so that they get a chance to release the service. Every bundle that gets notified is supposed to release the service by repeatedly calling `BundleContext.ungetService`, until its usage count of the service drops to zero.

For each bundle whose usage count of the service remains greater than zero after the bundle's service listener has returned (e.g., because the bundle has not properly released the service), the usage count will be set to zero; in addition, if the service was registered as a service factory, the `ServiceFactory.ungetService` method will be called to release the service object for the bundle.

4.6 Configurable Services

A service can be declared to be *configurable*. A configurable service is a service that can be configured dynamically at runtime, to change its behavior. For example, a configurable HTTP service may support a number of configuration options, including an option to set its port number.

A service is declared to be configurable by implementing the `Configurable` interface. This interface defines a single method, `getConfigurationObject`, which returns an object that holds the configuration data of the service and can be introspected. For example, the configuration object could be implemented as a JavaBean. The configuration object handles all the configuration aspects of a service. This way, a service does not have to expose its configuration properties itself.

The `getConfigurationObject` method is security-checked: Before returning the configuration object, it makes sure that the caller has the appropriate `AdminPermission`.

Chapter 5 *Events*

The Framework may generate three kinds of events:

- `ServiceEvent` - to report registration, unregistration, or property changes for services. All events of this kind are delivered synchronously.
- `BundleEvent` - to report changes in the lifecycle of bundles

- FrameworkEvent - to report that the Framework is started, or errors encountered by the Framework

For each kind of event, there is a corresponding listener interface:

- ServiceListener
- BundleListener
- FrameworkListener

and methods on the BundleContext class add and remove each type of listener.

Events are asynchronously delivered unless otherwise stated, meaning that the Framework does not wait for the listener methods to complete.

It is not defined which thread will be used to execute an event listener.

A security check is performed for each installed `ServiceListener` when a `ServiceEvent` occurs. The listener is not called unless it has permission to use the service.

A Bundle that uses services that could be unregistered should register a `ServiceListener` to monitor the availability of the services. It should take appropriate action when it finds out that the services have gone away.

Chapter 6 *Security*

6.1 Permissions

The security in Framework is based on the Java 2 specification. It is assumed that the reader is familiar with the document Java Security Architecture (JDK 1.2).

The Java platform that Framework runs on must provide the Java Security API necessary for Java 2 permissions. On very resource constrained platforms, these Java Security API may be stubs that allow the bundle classes to be loaded and executed but the stubs never actually perform the security checks. Alternatively, `AdminPermission`, `PackagePermission`, and `ServicePermission` classes can be left out of a Framework that does not support security. However, if the checks are performed, they must be done according to the Java Security Architecture (JDK 1.2).

Many of the methods of the Framework API require the caller to have certain permissions. Services may also have permissions specific to them that provide finer grained control over the operations that they can perform. Thus a bundle that exposes services to other bundles may also need to define permissions specific to the exposed services.

Normally when a permission check is done, the `java.security.AccessController` will check all the classes on the call stack to ensure that every one of them has the permission being checked. Since service methods often allow access to some resources to which only the bundle providing the service has access, a common programming pattern will use the `doPrivileged` method to convert a service permission into a resource access.

The following example is the dial method of the fictitious PPPService. In this example the dial method accesses the serial port to dial a remote server and start up the PPP daemon. The bundle providing the PPPService will have permission to execute programs and access the serial port, but the bundles using the PPPService 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. In this example it is the `org.osgi.service.ppp.DialPermission`. This check is done with the following segment of code:

```
    AccessController.checkPermission
       ( new org.osgi.service.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 follows:

```
    Process pppProcess = (Process)
        AccessController.doPrivileged(new PrivilegedAction()
            {
                 public Object run() {
                     Process pppProcess = null;
                     if ( connectToServer() ) {
                         pppProcess = startDaemon();
                     }
                     return pppProcess;
                 }

            }
        );
```

The preceding was an example of one way of executing privileged code. See the Java Security Architecture (JDK 1.2) document for more information.

The following permissions are defined by the Framework:

`AdminPermission`

This permission enables access to the administrative functions of Framework. It has no parameters associated with it. It enables access to the lifecycle management functions of Framework.

`ServicePermission`

The `ServicePermission` controls service registration and access. The permission has two parameters. The first parameter is an interface name. The interface name may end with a wildcard to match multiple interface names. (See `java.security.BasicPermission` for a discussion of wildcards.) The second parameter is the action. Supported actions are `register` and `get`. The `register` action indicates that the permission holder may register the service. The `get` action indicates that the holder may get the service.

The following actions require an appropriate `ServicePermission`:

- When an object is being registered as a service (using `BundleContext.registerService`), the registering code must have the `ServicePermission` to register *all* the named classes.
- When a `ServiceReference` is obtained from the service registry (using `BundleContext.getServiceReference` or `BundleContext.getServiceReferences`, respectively), the calling code must have the `ServicePermission` to get the service with the named class.
- When a service object is obtained from a `ServiceReference` using `BundleContext.getService`, the calling code must have the `ServicePermission` to get the service for at least one of the classes under which it was registered.

The permission is also used as a filter for the service events generated by the Framework as well as the enumeration methods to enumerate services (including `Bundle.getRegisteredServices` and `Bundle.getServicesInUse`). In general, a bundle will not be able to detect the presence of a service that it does not have permission to access.

`PackagePermission`

Bundles can only import and export packages for which they have permissions. A Package permission is given over all versions of a package. The `PackagePermission` has two parameters. The first parameter is the name of the package that may be exported. A wildcard may be used. Note that the granularity of the permission is the package, not the class name. The second parameter is either `import` or `export`. If a bundle has permission to export a package, Framework will automatically grant it permission to import the package. A `PackagePermission` with `*` and `export` as parameters would be able to import and export any package.

6.2 Policy

It is the responsibility of the gateway operator to configure the gateway's system policy in such a manner that bundles intended to be installed and run in the gateway are granted the required permissions to run properly.

Policies may be specified as files. See section 3.3.1 of the Java Security Architecture (JDK 1.2) for a policy file syntax.

OSGi Service Gateway 1.0: Device Access Specification

Contents

Chapter 1 *Introduction*

This specification describes the OSGi device access system. The device access system supports automatic detection of attached and detached hardware devices and can automatically download and start appropriate device drivers. Devices can be plugged and unplugged at any time and the device access system immediately responds to these changes.

The device access system is an optional component of OSGi.

This document assumes familiarity with the OSGi Services Framework, the overall structure of OSGi and the Java programming language.

1.1 Status

This is a draft version of the specification.

1.2 Goals

The OSGi architecture consists of a number of components (bundles) executing in a service framework. The target environments for this architecture have a number of specific characteristics, which the device access system must support:

- **Embedded devices**
  OSGi systems will run in embedded devices, meaning limited possibility for user interaction. Low cost systems also have limited resources.
- **Remote administration**
  OSGi platforms must support administration by a remote service provider.
- **Vendor neutrality**
  Individual components will be supplied by different vendors. Each component should be well-defined and replaceable.
- **Long running**
  OSGi systems will be running for extended periods (months, maybe years) without restarting, thus requiring stable operation and stable resource consumption.
- **Dynamic update**
  Components should as far as possible be individually replaceable without affecting unrelated components. Component updates should in particular not require restarting the whole system. In the model presented here, the device manager and driver locator bundles can be updated without disrupting operation of any connected devices.

Additionally, the device access system has a number of goals of its own:

- **Hot-plugging**
  The device access system must support plugging and unplugging of devices at any time
- **Automatic detection**
  The device access system should automatically detect plugged and unplugged devices whenever the underlying hardware allows it
- **Legacy support**
  On the other end of the scale, the device access system must also cope with device technologies that do not support any automatic detection
- **Dynamic driver loading**
  The device access system must be capable of loading new device drivers on demand with no prior device specific knowledge
- **Multiple device representations**
  The device access system must allow a device to be accessed from multiple levels of abstraction.
- **Deep trees**
  The device access system should allow connection of devices in a tree of mixed network technologies of arbitrary depth
- **Topology independence**
  The device access system must clearly separate the interface of a device from where and how it is attached
- **Complex devices**
  The device access system must support multifunction devices and devices that have multiple configurations

Chapter 2 *Concepts*

The OSGi device access architecture provides a framework to allow bundles within an OSGi environment to have access to physical devices.

Any physical device can be represented in several ways. For example, consider a USB mouse. It can be thought of as as a USB device which delivers information over the USB bus, or it can be thought of in a more refined manner as a mouse which delivers an x and y coordinate and information about whether the mouse's buttons are depressed. Each of these representations of the device has particular uses. A program which provides generic management of devices on a USB bus would not care whether the device was a mouse or not, while another program might only care about mouse-like input and not care whether the mouse was attached to a USB bus or a serial port. The OSGi device access architecture provides a means for a single device to be represented by one or more relevant representations within the OSGi Framework.

It may often be desirable for a user to plug a physical device into their home gateway and have the appropriate drivers automatically downloaded and installed so she may immediately use the new device. The OSGi device access architecture also provides this dynamic discovery and downloading of device drivers.

There are three entities which provide the multiple representations of a given physical device as well as the the dynamic discovery and downloading of device drivers: drivers, the device manager, and `DriverLocator` services.

A **driver** is responsible for providing a representation of a physical device within the OSGi Framework. A driver does this by registering a service which implements the `Device` interface with the Framework.

It can be convenient to think of drivers as belonging to one of two categories. The first category of drivers discover physical devices using software outside of the OSGi Device Access architecture (e.g. through notifications from a device driver in native code) and then register corresponding `Device` services. These are sometimes called *base drivers* since they provide the lowest-level representation of a physical device. An example of this category of driver would be a USB driver that discovers the presence of a USB device (e.g. a mouse) and registers it with the framework as a generic USB device.

The second category of drivers provide a refined view of a physical device that is already represented by another `Device` service registered with the Framework. These are sometimes called *refining drivers*. These drivers register a `Driver` service with the Framework which the device manager uses to attach the refining driver to a less refined `Device` service. An example of this category would be a mouse driver which is attached to the generic USB representation of a mouse and then registers a `Device` service which represents the physical device as a mouse.

In the case of base drivers, `Device`s are registered as a result of events outside the scope of the framework, while in the case of refining drivers, `Device`s are registered as a result of events within the framework itself.

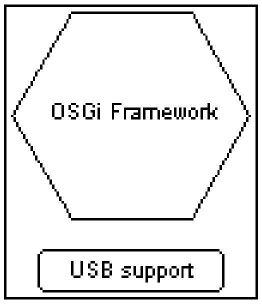
Note: This specification uses the term "`Driver`" (note the capital letter) to refer to a service implementing the `Driver` interface, while it uses the term "driver" to refer to the entity described immediately above. Similarly, it uses the term "`Device`" to refer to a service implementing the `Device` interface, while it uses "device" to refer to a physical device.

The **device manager** coordinates the relationships between certain drivers so that they can present multiple representations of the same device, and initiates the process of downloading new drivers.

`DriverLocator` services are where vendor specific knowledge about the location of drivers is located. The device manager uses `DriverLocator` services to identify and download new drivers when they are needed.

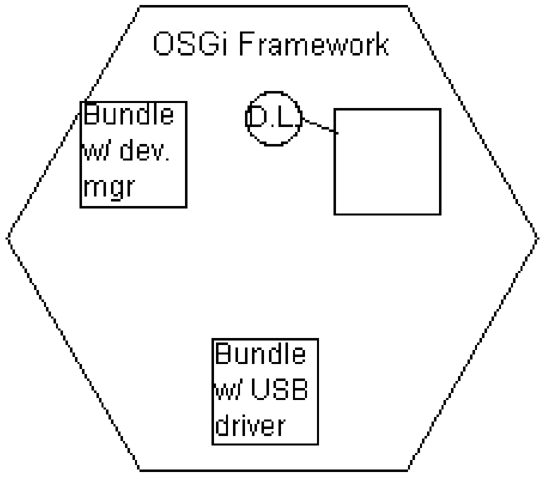
2.1 An Example

Consider a situation where an OSGi Framework is running on a system where there is support for the USB standard outside of the OSGi Framework:



When the device provider delivers this environment to a customer, it will also include:

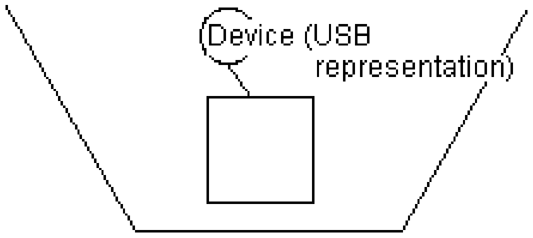
- a device manager (generally delivered in its own bundle)
- one or more bundles that will register `DriverLocator` services
- at least one driver which uses the underlying USB support.



Now, consider this sequence.

1) A mouse is attached to the USB bus of the OSGi device.

2) The USB driver discovers the mouse and registers a corresponding `Device` service. Note that this service only represents the mouse as a generic USB device. That is, the `Device` service will have no methods that are specific to mice.



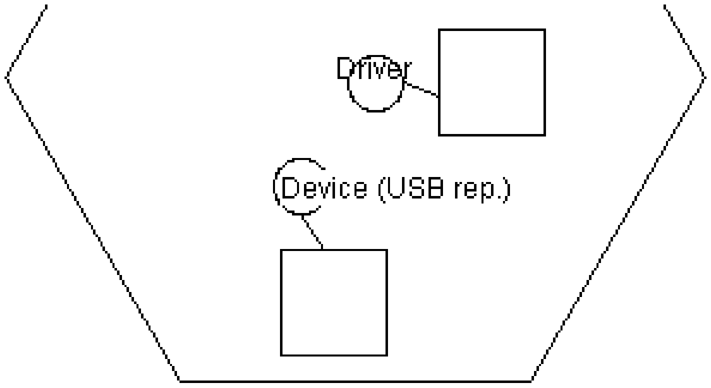
2) The device manager is notified of the newly registered `Device`. The device manager now starts a sequence that has the purpose of locating any drivers which might be able to provide a refined representation of the device (i.e. to

present it as a mouse)

3) The device manager asks the `DriverLocator` service for the DRIVER_ID's of any `Driver`s that might be able to present a refined representation of the `Device` service. In this example, the `DriverLocator` service returns a single ID.

4) The device manager then asks the `DriverLocator` service to download the bundles which can provide `Driver`s with the specified DRIVER_ID's. As each is downloaded, the device manager installs and starts each. In this example, a single bundle is downloaded, installed and started.

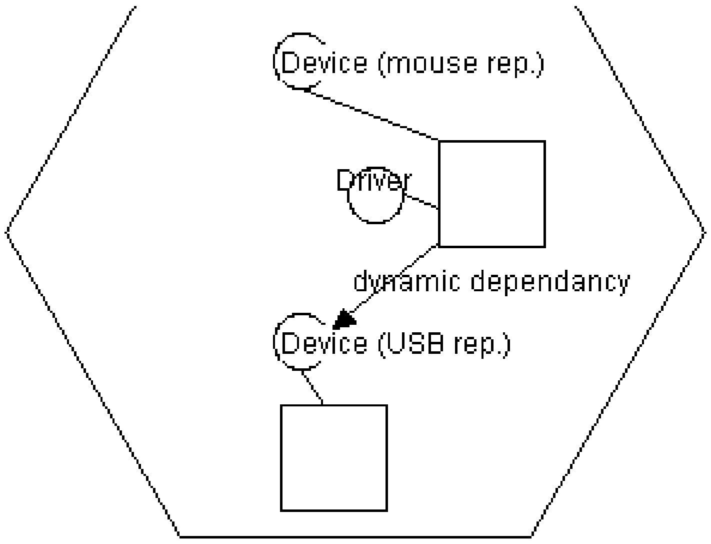
5) The new bundle registers a `Driver` service when it is started. This service is used by the device manager to interact with the driver.



6) The device manager is notified of the registration of the `Driver` service, and so it asks the `Driver` service how well it can match (that is, make use of) the `Device` service which had been registered earlier. In this example, the `Driver` returns a value indicating that it can use the `Device` service.

7) The device manager tells the `Driver` to try to attach to the `Device`. If this succeeds (which it does in this example) a dependency is established between the `Driver`'s bundle and the `Device` service.

8) When the attach operation is completed, the new driver registers a new `Device` service. The registration of this `Device` service causes the device manager to start looking for another driver to present a still more refined representation of the `Device`. In this example, the `DriverLocator` services return no DRIVER_ID's and there are no other `Driver`'s registered with the Framework, so the sequence started in step 2 stops.



Thus the physical mouse device is represented by two `Device` services. The bottom one represents the mouse in a generic USB form, while the upper represents it specifically as a mouse. The USB driver discovers new physical devices by interacting with software support outside of the OSGi environment. The upper driver has registered a

`Driver` service which the device manager uses to build the sequence of gradually more refined representations of the same physical device.

Chapter 3 *Details of Device Access*

This chapter presents full details about all the bundles and services that make up the OSGi Device Access Architecture.

3.1 The Device Manager

The device manager provides two related functions that are essential to the functioning of the OSGi Device Access architecture:

- It attaches `Driver` services from drivers to `Device` services, thereby producing multiple representations of the same device.
- It is responsible for working with `DriverLocator` services to dynamically download and install new drivers

Every OSGi device that supports the OSGi Device Access architecture must have exactly one device manager installed. Generally it will be contained in its own bundle. Unlike many bundles, the device manager installs no services. Instead, it monitors the OSGi framework's registry for the registration of `Device` services.

When a `Device` service is registered by a driver, the device manager will perform the following algorithm:

1. `DriverLocator.findDrivers()` will be called on any registered `DriverLocator` service passing the properties that were used to register the new `Device` service. Each method call will return zero or more DRIVER_ID values (identifiers of particular `Driver` services).
2. After excluding any DRIVER_ID's that correspond to `Driver` services that are already registered with the Framework registry, the device manager will call `DriverLocator.loadDriver()` to dynamically download, install and start a bundle for each remaining DRIVER_ID. Note that each bundle will register a `Driver` service when it is started.
3. `Driver.match()` will be called on every registered `Driver` service, and the ServiceReference corresponding to the newly registered `Device` will be passed to the `Driver` service.
4. If `Device.MATCH_NONE` is returned by all the `Driver` services, the device manager will call `Device.noDriverFound()` and skip to step 7.
    - Note that this may cause the `Device` to unregister its device and register a new one, which will cause the device manager to start this algorithm again.
5. Otherwise, the device manager will call `Driver.attach()` on the `Driver` service which returned the highest number from `match()`. If multiple `Driver` services returned the same highest number, the device manager is free to arbitrarily select the `Driver` to call `attach()` on`.`
6. `If attach()` returns a non-null value, this `Driver` is said to be *referring* the device manager to another `Driver`. The return value is assumed to be the DRIVER_ID of a different `Driver` capable of attaching to the specified `Device` service. If the `Driver` is not already installed and started, the device manager will then obtain the corresponding bundle from a `DriverLocator`, install it. The device manager continues by going to step 3 of this algorithm (though, the `Driver` that returned the non-null value from `attach()` will not be consulted again for the remainder of this algorithm)
7. If any device driver bundles were installed in step 2 or 6 but not used afterwards, the device manager is free to uninstall them.

The device manager is free to uninstall any bundles containing drivers that it earlier installed that are not attached to any `Device`s. In order to do this, the developer of the device manager must make a tradeoff between wasting resources by keeping unused bundles installed and the risk of uninstalling a bundle that will be needed again.

The device manager may cache the return value from `match()` to prevent unnecessary calls to this method.

The device manager must not base its actions on persistent records. That is, it should identify when `Drivers` have been attached to `Devices` by examining the dependencies between bundles and services.

3.2 `DriverLocator` Services

`DriverLocator` services provide the mechanism for dynamically downloading new drivers into an OSGi device. These services are supplied by OSGi providers. These encapsulate all provider-specific details related to the location and acquisition of bundles containing drivers, which allows other components of the device architecture to be developed in a provider-neutral fashion.

`DriverLocator` services must implement this interface:

```
public interface DriverLocator {
    public String[] findDrivers(Dictionary prop);
    public InputStream loadDriver(String driverId) throws IOException;
}
```

The `findDrivers()` method is passed a Dictionary of properties and returns an array of Strings representing the DRIVER_ID's of a set of `Driver`s which may be able to attach to the `Device`. `DriverLocator` services should try to return as few DRIVER_ID's as possible, since each corresponding bundle that is not already installed will be downloaded, and in some cases this may be done over a fairly slow communications line. Therefore, the `DriverLocator` might use the following order of preference when selecting the DRIVER_ID's to return:

- the only or "best" driver
- a driver which is known to be able to provide a reference to a better driver
- multiple possible drivers (excluding any which will return lower `match()` return values than others in the set)

The `loadDriver()` method is used to obtain a new device driver bundle, given an ID.

Both of these methods should only be called by the device manager.

3.3 Drivers

A driver exists to present a particular representation of a category of devices, which it accomplishes by registering a `Device` service. `Device` registration can be triggered in two ways. In the first case, a driver "discovers " a new physical device by interacting with software outside the OSGi environment and so registers a corresponding `Device` service. While these drivers often correspond to some kind of network (e.g. a USB bus, a serial port, etc.) which a physical device may be attached to, they may also correspond to a non-network entity, such as a display. In the second case, the driver registers a `Device` service which is a refined representation of an existing `Device` service. A driver that presents a refined representation of an existing `Device` must have registered a `Driver` service, which the device manager attaches the existing `Device` to, which causes the driver to register the new `Device`.

Note that some drivers may be able to represent a physical device in several ways. When this is the case, the driver should first register a `Device` service which represents the best, or most comprehensive, representation of the device. If `noDriverFound()` is called on that `Device` service, the driver may register an alternate `Device` service (or services) which represent the physical device in a different way.

When a driver is unable to automatically detect when new physical devices have become available, it should provide some custom mechanism for another entity (e.g. an end user) to notify it that a device is available.

When a driver detects (or is otherwise informed) that a physical device has ceased to be available, it should unregister the corresponding `Device` service(s). Any bundles which have been attached to these services should release their dependencies on the services and withdraw any refined representation(s) of the physical device. This will result in the removal of all representations of the physical device.

It is also permissible to have a driver which registers a `Driver` service and will never register any `Device` services nor will it attach to any `Device` services. In this case, the driver only exists to provide referrals to other drivers through the return value from its `attach()` method.

Drivers should never establish a dependency on a `Device` service for any reason except as the result of a call to `Driver.attach()` which should only be called by the device manager.

3.3.1 `Driver` Services

```
public interface Driver {
    public int match(ServiceReference sr) throws Exception;
    public String attach(ServiceReference SR) throws Exception;
}
```

Drivers must register their `Driver` services in BundleActivator.start(), since if they do it later on, they may miss out being available to be attached to new `Device`s.

match() checks whether the `Driver` can be attached to the `Device` specified by the ServiceReference. This returns a code which it gets from the interface of the kind(s) of `Device`s that it can attach to. While a `Driver` may simply examine the properties associated with the service to make its decision, it is free to get the service object itself, as long as it ungets it and returns the physical device to a normal state before it returns. Note that the device manager may cache the return value, and so `Driver`s should always return the same result code whenever presented with the same service.

`attach()` tells the driver to establish a dynamic dependency on the specified service. If it is able to do this, it returns null. Otherwise, it returns a DRIVER_ID value which acts as a referral to a `Driver` that can make better use of the specified `Device`. If the `Driver` can neither attach to the `Device` nor offer a referral, it will throw an exception, however this can only happen if it mistakenly returned a legitimate value from the `match()` method. The device manager will only call this method if a previous call to `match()` returned a value larger than `Device.MATCH_NONE`.

These methods should only be called by the device manager.

Every `Driver` service must have an associated property named "DRIVER_ID" whose value is a string identifying it. This string should start with the reversed form of the domain name of the company that implemented it (e.g. com.acme). This ID is used by the device manager to prevent duplicate copies of a driver from being installed. Consequently, the DRIVER_ID should have the following properties:

- it must be independent of the location that it is obtained from
- it must be independent of the `DriverLocator` service used to download it
- it must be different from the ID of different drivers
- it must be different for different revisions of the same driver

3.3.2 Device Services

```
public interface Device {
    public static final int MATCH_NONE = 0;
```

```
        public void noDriverFound();
    }
```

The `MATCH_NONE` value is returned by `Driver.match()` if the `Driver` service can not attach to the `Device` service that it is passed.

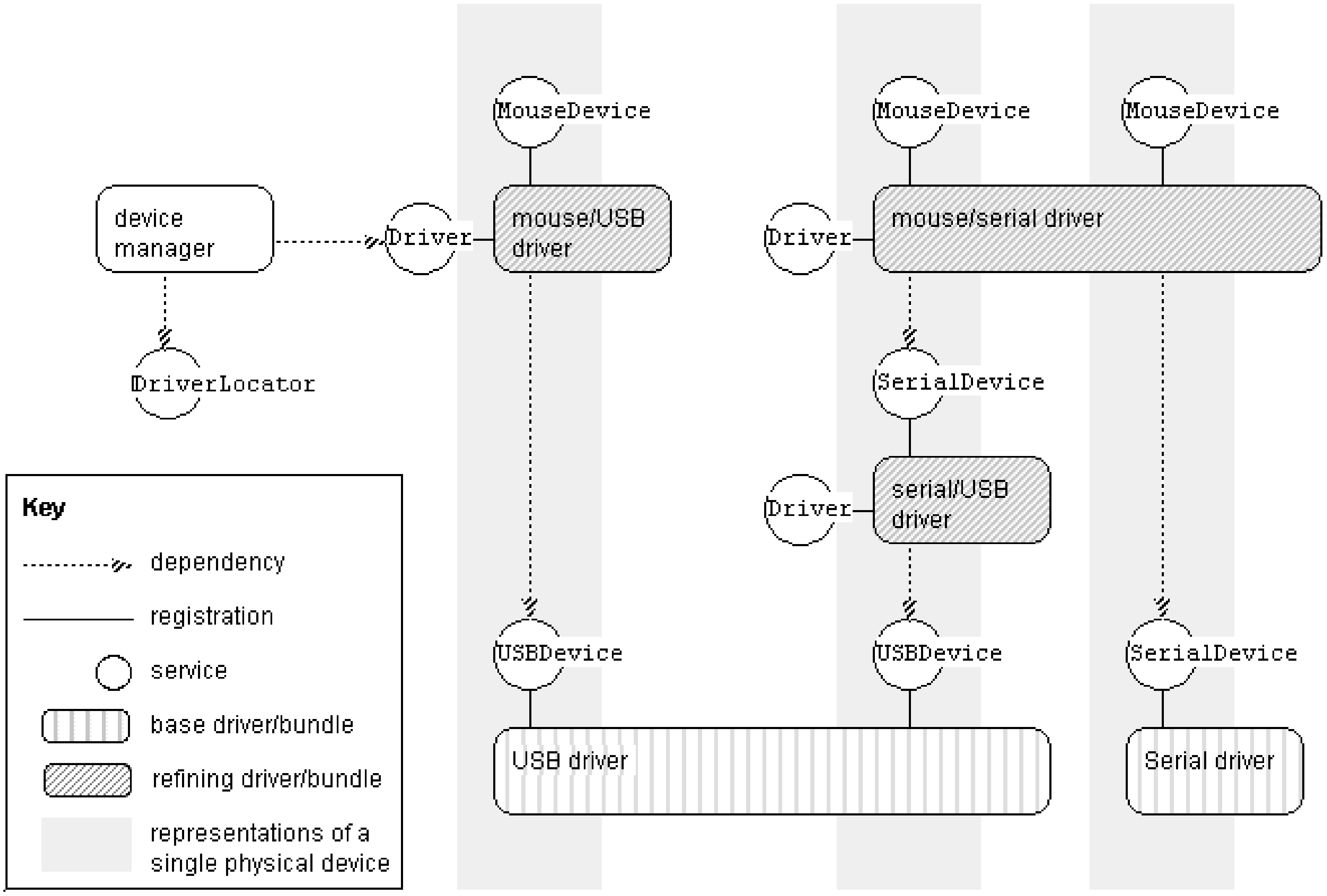
`noDriverFound()` is called by the device manager when it can not locate a `Driver` to attach to the device. This may prompt the driver to present a new `Device` service which represents the device differently.

Since a single physical device may be represented by multiple `Device` services, implementers may sometimes need to provide a mechanism to prevent other bundles from writing to or reading from the device through more than one of the `Device` services simultaneously. This is up to the implementer to coordinate if it is needed.

`Devices` should be registered with multiple class names. One of these names must be `org.osgi.service.device.Device` so the device manager can identify which services are `Devices`.

It is expected that future versions of the OSGi specification will include subclasses of the `Device` service interface, and corresponding sets of properties and match constants, which are specialized for different situations.

Chapter 4 *Examples*



The figure above illustrates several points:

- This represents three physical devices:
    - a USB mouse (represented as a generic `USBDevice` and as a `MouseDevice`)

  - a serial mouse attached to a serial to USB bridge (represented as a generic `USBDevice`, a `SerialDevice` and a `MouseDevice`)
  - a serial mouse attached to a serial port (represented as a `SerialDevice` and a `MouseDevice`)
- Base drivers Serial and USB do not have any `Driver` services and are preinstalled.
- Refining drivers mouse/USB, mouse/Serial and serial/USB have registered one `Driver` service each, and have probably been installed by the device manager (though they could also have been preinstalled).
- There is presently one `DriverLocator` service.
- All three `MouseDevice`s have the same interface even though they are built on different technologies and are attached at different places.
- The mouse/serial driver is attached to two serial "things" which it provides refinements of as mice.
- The device manager is currently setting up the mouse/USB driver since it has a dependency on its `Driver` service.

Chapter 5 *Device categories*

OSGi will define a number of device categories. For each category there will be an interface definition approved by OSGi. All `Device`s in a defined category must conform to the approved interface, thus ensuring interoperability between `Device`s from different vendors. A category interface definition consists of:

- An extension of `Device`, with:
  - Methods appropriate to the category.
  - A match value scale appropriate to the category.
- A set of properties to accompany device service registrations. All categories must include `DEVICE_CATEGORY` as a mandatory property. If `DEVICE_SERIAL` is used, it must be unique for each device.

5.1 Examples of categories

- USB
- IEEE1394
- Mouse
- Keyboard

5.2 An incomplete category definition example

`Device` interface:

```
interface USBDevice extends Device {
  static final int MATCH_CLASS_MAKE_MODEL_REV_SERIAL = 6;
  static final int MATCH_CLASS_MAKE_MODEL_REV        = 5;
  static final int MATCH_CLASS_MAKE_MODEL            = 4;
  static final int MATCH_CLASS_MAKE                  = 3;
  static final int MATCH_CLASS                       = 2;
  static final int MATCH_GENERIC                     = 1;

  void sendPacket(Pipe pipe, Data data);
```

```
    Data receivePacket(Pipe pipe);
    ...
  }
```

`Device` properties:

```
  DEVICE_CATEGORY   mandatory  = "USB"
  DEVICE_CLASS      mandatory
  DEVICE_MAKE       mandatory
  DEVICE_MODEL      mandatory
  DEVICE_REVISION   optional
  DEVICE_SERIAL     optional
```

Chapter 6 *Security*

All installation and uninstallation of drivers is done by the device manager since the `DriverLocator`s will not have generally been granted the required AdminPermission.

OSGi Service Gateway
Release 1.0 Final
All Classes

Packages
org.osgi.framework
org.osgi.service.device
org.osgi.service.http
org.osgi.service.log

All Classes

AdminPermission
Bundle
BundleActivator
BundleContext
BundleEvent
BundleException
BundleListener
Configurable
Device
Driver
DriverLocator
FrameworkEvent
FrameworkListener
HttpContext
HttpService
InvalidSyntaxException
LogEntry
LogListener
LogReaderService
LogService
NamespaceException
PackagePermission
ServiceEvent
ServiceFactory
ServiceListener
ServicePermission
ServiceReference
ServiceRegistration

OSGi Service Gateway API Specification Release 1.0 Final

Framework	
org.osgi.framework	The OSGi Java Services Framework.

Services	
org.osgi.service.device	The OSGi Device Access Specification.
org.osgi.service.http	The OSGi HttpService Specification.
org.osgi.service.log	The OSGi LogService Specification.

Package org.osgi.framework

The OSGi Java Services Framework.

See:
Description

Interface Summary

Bundle	A bundle installed in a framework.
BundleActivator	Customizes the starting and stopping of a bundle.
BundleContext	Bundle's execution context.
BundleListener	BundleEvent listener.
Configurable	Interface implemented by services which support a configuration object.
FrameworkListener	FrameworkEvent listener.
ServiceFactory	Service factories allow services to provide customized service objects.
ServiceListener	ServiceEvent listener.
ServiceReference	A reference to a service.
ServiceRegistration	A registered service.

Class Summary

AdminPermission	The AdminPermission indicates the caller's right to perform life-cycle operations on or to get sensitive information about a bundle.
BundleEvent	Bundle lifecycle change event.
FrameworkEvent	General framework event.
PackagePermission	PackagePermission indicates a bundle's right to import or export a package.
ServiceEvent	Service lifecycle change event.
ServicePermission	ServicePermission indicates a bundle's right to register or get a service.

Exception Summary

BundleException	Exception from the framework to indicate a bundle lifecycle problem occured.

InvalidSyntaxException	Exception from the framework to indicate a filter string parameter has an invalid syntax and cannot be parsed.

Package org.osgi.framework Description

The OSGi Java Services Framework.

OSGi

Open Services Gateway Initiative

Welcome to the Open Service Gateway Initiative (OSGi) Framework API. This page gives an overview of what this framework is intended for and what the overall structure is.

The purpose of the framework is to provide a context for application developers to write code for small devices that are continuously running. In these environments applications are swapped in and out, they are updated on the fly and they must communicate in a structured and dependable way with other applications. Obviously this is quite different from normal applications that are started from a command line or mouse click.

The OSGi Framework takes advantage of the Java™ programming language's ability to download code from the network. It provides a rich and structured development platform for component-based architectures.

Goals

The primary goal of the framework is to provide an environment that supports:

1. Dynamic load or update applications on the fly without stopping the environment.
2. Be able to use in limited memory devices
3. Offer a concise and consistent component programming model for application developers
4. Manage dependencies between applications
5. Scalable.

The OSGi framework provides a life-cycle management framework that permits application developers to partition applications into small self-installable applications. These applications are called bundles in this context. Devices running the framework can download bundles on demand and remove them when they are no longer required. A bundle, when installed can register any number of services to be shared with other bundles under control of the framework.

The OSGi Framework also supports application developers in coping with scalability issues. This support is critical, because the framework is designed to run in a variety of devices; the different hardware characteristics of the devices could affect the scale of the services that they are able to support. The framework supports the development and use of services of varying scale by decoupling a service's specification from its implementation. By splitting the implementation of a service from its specification (its Java interface):

- Developers of service implementations can implement the same interface
- Developers who use a service can code against that service's interface without regard to its implementation

For example, in a high-end device, a logging service might be able to store log messages on a hard drive, while on a disk-less device, the log entries may have to be saved remotely. The developers of the two logging service implementations implement the same interface; the developers of services that use a logging service write code against the logging service interface, without regard to which implementation their service might use. The framework can fully hide and manage the different implementations from the bundles that use the service.

Concepts

The OSGi Framework is a lightweight framework for creating extensible services using the Java programming language. To take best advantage of the framework, developers should design an application as a set of services, with each service implementing a segment of the overall functionality. These services and other extension services are then packaged in a "bundle" and downloaded on demand by the target device. For example, a text editing application designed this way could rely on a spell-checking service. The editor would instruct its framework to download a spell-checker for it to use. The framework provides mechanisms to support this paradigm that are simple to use and help with the practical aspects of writing extensible applications. The key entities of the framework are:

- Services - Objects that provide a collection of methods providing some service
- Bundles - The infrastructure for delivering code
- Bundle contexts - The execution environments of the bundles

Services

In the OSGi Framework model, an application is built around a set of cooperating services: it can extend itself at runtime by requesting the services it requires. The framework maintains a set of mappings from services to their implementations and has a powerful query mechanism that enables an installed bundle to request and use the available services. The framework manages dependencies between services and bundles. E.g. when a bundle is stopped, all the services that it registered will be unregistered automatically.

A standard group like the OSGi or a private group of developers define a service as an interface and publish it. Any developer can now offer this service by implementing this standard interface. Other developers can now write code that uses this interface and obtain implementations via the framework.

The framework provides an API that the developer then uses to register the service. A service can be registered with an optional set of properties describing the service. This could for example be the name of manufacturer, version, interfaces or author. Any bundle that now wants to use a service can specify a filter string that can be used to filter the service registry for available services. The framework can be asked to pick any service that fullfils the filter or it can return a list of services.

The framework hands out references to services. These references can be queried for properties and other meta information. A reference to a desired service can be used to obtain the service object which implements the service. The framework tracks the services that are being used by each bundle.

Bundles

To be available to the framework, a service implementation must be packaged. Service implementations are packaged into entities called bundles. A bundle is a JAR file that:

- Contains the resources implementing zero or more services. These resources may be class files for the Java programming language, as well as any other data (such as HTML help files, icons, and so on).
- States static dependencies on some other resources, such as Java packages. If any dependencies are stated, the framework takes the appropriate actions to make the required resource available.

- Manifest header describing which class should be used to start() or stop() a service.

By packing all these items together in a single JAR file, the framework can uniformly download and control a bundle.

To write a bundle, the developer must put a tag in the manifest that defines the name of a class that implements the `BundleActivator` interface. This interface has a start and stop method to start/stop the bundle. In the start method the bundle should wait until all its requirements are fullfilled and then start registering its services or fullfilling its task.

Bundle Manifest Headers

The framework recognizes special headers in the bundle's manifest. These headers and their values contain important information about how the bundle will operate in the framework. Additional headers can be defined by a bundle programmer as needed. Headers and their values of all manifest headers can be retrieved with `Bundle.getHeaders()` method.

The following headers in the bundle's manifest have special meaning to the framework.

`Bundle-Activator:` *`class-name`*

If present, this header names the class in the bundle which implements the `BundleActivator` interface. This class will used when the bundle is started and stopped. *`class-name`* must be a fully qualified Java class name.

`Bundle-Name:` *`string`*

If present, this header contains a human readable string which is the name of the bundle. This information is available at runtime using the bundle's `Bundle.getHeaders()` method. The name string is the value of the key "Bundle-Name".

`Bundle-Vendor:` *`string`*

If present, this header contains a human readable string describing the vendor of the bundle. This information is available at runtime using the bundle's `Bundle.getHeaders()` method. The vendor string is the value of the key "Bundle-Vendor".

`Bundle-Version:` *`string`*

If present, this header contains a human readable string which is the version of the bundle. This information is available at runtime using the bundle's `Bundle.getHeaders()` method. The version string is the value of the key "Bundle-Version".

`Bundle-Description:` *`string`*

If present, this header contains a human readable string describing the purpose or function of the bundle. This information is available at runtime using the bundle's `Bundle.getHeaders()` method. The description string is the value of the key "Bundle-Description".

`Bundle-DocURL:` *`string`*

If present, this header contains a human readable string which is a URL which contains further information about the bundle. This information is available at runtime using the bundle's `Bundle.getHeaders()` method. The doc URL string is the value of the key "Bundle-DocURL".

`Bundle-ContactAddress:` *`string`*

If present, this header contains a human readable string which is an e-mail address which can be contacted regarding the bundle. This information is available at runtime using the bundle's `Bundle.getHeaders()` method. The contact address string is the value of the key

"Bundle-ContactAddress".

`Bundle-UpdateLocation:` *`string`*

If present, this header contains a location string that will be used to locate an updated version of the bundle when the bundle is updated using the `Bundle.update()` method. If this header is not present, the bundle will be updated from its original location. The location string will typically be a URL.

`Bundle-ClassPath:` *`path (, path )*`*

If present, this header describes the classpath within the bundle. A bundle's JAR file can contain other JAR files within it. So the Bundle-ClassPath is used to describe the search order for classes. *`path`* can be either *dot* ('.') which represents the bundle's JAR file or it can be the path of a JAR file contained in the bundle's JAR file. If Bundle-ClassPath is not specified, the default value is *dot*. If Bundle-ClassPath is specified, but *dot* is not an element of the path, then the bundle's main JAR file will not be searched. Only the contained JAR files referenced by the path elements will be searched.

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

If present, this header describes the packages which the bundle offers for share with other bundles. *`package-description`* has the following format

`package-name (;` `specification-version=`*`version-number )`*

`package-name` is the fully qualifed name of a Java package. *`version-number`* is the version number of the package as specified by Java™ Product Versioning Specification. Since multiple bundles may offer to share the same package, the framework will select the bundle offering to share the package at the highest version. The framework guarantees that only one bundle will be selected to share a given package name.

A bundle which offers to export a package must also import that package. If the bundle's manifest does not have an Import-Package header for the same package name, the bundle will automatically import the package using the same *`package-description`* it has offered for export.

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

If present, this header describes the packages which the bundle requires. These packages must be exported by another bundle. See Export-Package for the definition of *`package-description`*. If no bundles export the required packages, then this bundle is not permitted to offer packages for export.

Packages that are part of the Java platform, such as those package names starting with "java." should not be referenced in the Import-Package header. For all other packages required by the bundle, the package names of those packages should be specified in the Import-Package header.

`Export-Service:` *`class-name (, class-name )*`*

If present, this header describes the services the bundle may register. This header provides advisory information that is not used by the framework. It is intended for use by server-side management tools.

`Import-Service:` *`class-name (, class-name )*`*

If present, this header describes the services the bundle may use. This header provides advisory information that is not used by the framework. It is intended for use by server-side management tools.

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

If present, this header describes the files which contain the implementation of the native methods used by classes in the bundle. *`nativecode-clause`* has the following format

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

```
env-parameter: ( processordef | osnamedef | osversiondef | languagedef )
processordef: processor=token
osnamedef: osname=token
osversiondef: osversion=token
languagedef: language=token
```

`nativepath` is the path of a file in the bundle's JAR file containing native methods. The `env-parameter`s are compared against property values provided by `BundleContext.getProperty()`.

- `processor` is compared against `org.osgi.framework.processor`.
- `osname` is compared against `org.osgi.framework.os.name`.
- `osversion` is compared against `org.osgi.framework.os.version`.
- `language` is compared against `org.osgi.framework.language`.

If a group of specified `env-parameter`s match the property values, the framework will make the files specified by the `nativepath`s preceding the `env-parameter`s available to be loaded by `System.loadLibrary`.

Care should be taken to select a unique name for the file containing the native methods. The framework will extract the selected files from the bundle into a location where `System.loadLibrary` can find them. This location will be shared by all bundles.

When the framework starts and stops

When the framework is started. The following actions occur:

1. Event handling is enabled. Events can now be delivered to listeners.
2. The Framework persistently records whether an installed bundle has been started or stopped. When the framework is restarted, all installed bundles previously recorded as being active will be automatically started as described in the `Bundle.start` method. Reports any exceptions that occur during startup using `FrameworkEvents`.
3. A `FrameworkEvent` of type `STARTED` is broadcast.

When the framework is stopped. The following actions occur:

1. Suspend all active bundles as described in the `Bundle.stop` method except that their persistently recorded states remain to be ACTIVE. These bundles will be restarted when the framework is next started. Reports any exceptions that occur during stopping using `FrameworkEvents`.
2. Event handling is disabled.

Conformance Statement

Conforming implementations must not add any new methods or fields to any of the classes or interfaces defined in the OSGi specification (though, they may add them to subclasses, or classes that implement the interfaces), nor may they add any new classes or packages to the org.osgi package tree.

OSGi Service Gateway
Release 1.0 Final

org.osgi.framework

Interface ServiceRegistration

public interface **ServiceRegistration**

A registered service. The framework returns a ServiceRegistration object when a `BundleContext.registerService` method is successful. This 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 for the service or to unregister the service.

If the ServiceRegistration is garbage collected the framework may remove the service. This implies that if a bundle wants to keep its service registered, it should keep the ServiceRegistration object referenced.

Version:
 1.0

Author:
 Open Services Gateway Initiative

Method Summary

`ServiceReference`	**`getReference`**`()` Returns a `ServiceReference` object for this registration.
`void`	**`setProperties`**`(java.util.Dictionary properties)` Update the properties associated with this service.
`void`	**`unregister`**`()` Unregister the service.

Method Detail

getReference

`public ServiceReference` **`getReference`**`()`

Returns a `ServiceReference` object for this registration. The `ServiceReference` object may be shared with other bundles.

Returns:

A `ServiceReference` object.

Throws:

java.lang.IllegalStateException - If this ServiceRegistration has already been unregistered.

---

setProperties

```
public void setProperties(java.util.Dictionary properties)
```

Update the properties associated with this service.

The key "objectClass" cannot be modified by this method. It's value is set when the service is registered.

The following steps are followed to modify a service's properties:

1. The service's properties are replaced with the provided properties.
2. A `ServiceEvent` of type `ServiceEvent.MODIFIED` is synchronously sent.

Parameters:

`properties` - The properties for this service. Changes should not be made to this object after calling this method. To update the service's properties this method should be called again.

Throws:

java.lang.IllegalStateException - If this ServiceRegistration has already been unregistered.

---

unregister

```
public void unregister()
```

Unregister the service. Remove a service registration from the framework's service registry. All `ServiceReference` objects for this registration can no longer be used to interact with the service.

The following steps are followed to unregister a service:

1. The service is removed from the framework's service registry so that it may no longer be used. `ServiceReference`s for the service may no longer be used to get a service object for the service.
2. A `ServiceEvent` of type `ServiceEvent.UNREGISTERING` is synchronously sent so that bundles using this service may release their use of the service.
3. For each bundle whose use count for this service is greater than zero:
    1. The bundle's use count for this service is set to zero.

2. If the service was registered with a `ServiceFactory`, the `ServiceFactory.ungetService` method is called to release the service object for the bundle.

Throws:

java.lang.IllegalStateException - If this ServiceRegistration has already been unregistered.

See Also:

`BundleContext.ungetService(org.osgi.framework.ServiceReference)`

---

org.osgi.framework
Class BundleException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--org.osgi.framework.BundleException
```

---

public class **BundleException**
extends java.lang.Exception

Exception from the framework to indicate a bundle lifecycle problem occured. It is created by the framework to denote an exception condition in the lifecycle of a bundle. BundleExceptions should not be created by bundle programmers.

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
Serialized Form

Constructor Summary

`BundleException(java.lang.String msg)` Create a bundle exception with the given message.
`BundleException(java.lang.String msg, java.lang.Throwable throwable)` Create a bundle exception that wraps another exception.

Method Summary

`java.lang.Throwable`	`getNestedException()` Retrieve any nested exception included in this exception.

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

BundleException

```
public BundleException(java.lang.String msg,
                       java.lang.Throwable throwable)
```

Create a bundle exception that wraps another exception.

Parameters:

`msg` - The associated message.

`throwable` - The nested exception.

BundleException

```
public BundleException(java.lang.String msg)
```

Create a bundle exception with the given message.

Parameters:

`msg` - The message.

Method Detail

getNestedException

```
public java.lang.Throwable getNestedException()
```

Retrieve any nested exception included in this exception.

Returns:

The nested exception, or **null** if there is no nested exception.

OSGi Service Gateway
Release 1.0 Final

---

org.osgi.framework

Class InvalidSyntaxException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--org.osgi.framework.InvalidSyntaxException
```

public class **InvalidSyntaxException**
extends java.lang.Exception

Exception from the framework to indicate a filter string parameter has an invalid syntax and cannot be parsed.

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
Serialized Form

Constructor Summary

InvalidSyntaxException(java.lang.String msg, java.lang.String filter) Create the exception with the given message and the filter string which generated the exception.

Method Summary

java.lang.String	**getFilter**() Returns the filter string which generated the exception.

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,
wait, wait
```

Constructor Detail

InvalidSyntaxException

```
public InvalidSyntaxException(java.lang.String msg,
                              java.lang.String filter)
```

Create the exception with the given message and the filter string which generated the exception.

Parameters:

`msg` - The message.

`filter` - The invalid filter string.

Method Detail

getFilter

```
public java.lang.String getFilter()
```

Returns the filter string which generated the exception.

Returns:

The invalid filter string.

See Also:

`BundleContext.getServiceReferences(java.lang.String, java.lang.String)`, `BundleContext.addServiceListener(org.osgi.framework.ServiceListener, java.lang.String)`

org.osgi.framework

Interface BundleActivator

public interface **BundleActivator**

Customizes the starting and stopping of a bundle. BundleActivator is an interface that may be implemented by a bundle programmer that is called when the bundle is started or stopped. A bundle can only specify a single BundleActivator in the manifest.

When a bundle is started or stopped, the BundleActivator is called.

The framework can create instances of the bundle's BundleActivator as required. It is guaranteed, however, that if an instance's `start` method executes successfully, that same instance's `stop` method will be called when the bundle is to be stopped.

A bundle programmer specifies the BundleActivator through the `Bundle-Activator` Manifest header. The form of the header is:

```
 Bundle-Activator: class-name
```

where *`class-name`* is a fully qualified Java classname. The specified BundleActivator class must have a public constructor that takes no parameters so that a BundleActivator object can be created by `Class.newInstance()`.

Version:

1.0

Author:

Open Services Gateway Initiative

Method Summary

`void`	**`start`**(`BundleContext context`) Called when the bundle is started so that the bundle can perform any bundle specific activities to start the bundle.
`void`	**`stop`**(`BundleContext context`) Called when the bundle is stopped so that the bundle can perform any bundle specific activities necessary to stop the bundle.

Method Detail

start

```
public void start(BundleContext context)
           throws java.lang.Exception
```

Called when the bundle is started so that the bundle can perform any bundle specific activities to start the bundle. Bundle programmers can use this method to register the bundle's services or to allocate any resources that the bundle needs.

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

Parameters:

`context` - The execution context of the bundle being started.

Throws:

java.lang.Exception - If this method throws an exception, the bundle is marked as stopped and the framework will remove the bundle's listeners, unregister all service's registered by the bundle, release all service's used by the bundle.

See Also:

`Bundle.start()`

---

stop

```
public void stop(BundleContext context)
          throws java.lang.Exception
```

Called when the bundle is stopped so that the bundle can perform any bundle specific activities necessary to stop the bundle. In general, this method should undo the work that the `start` method did. When this method returns the bundle should have no active threads. A stopped bundle should be stopped and should not be calling any framework objects.

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

Parameters:

`context` - The execution context of the bundle being stopped.

Throws:

java.lang.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 service's registered by the bundle, release all service's used by the bundle.

See Also:

`Bundle.stop()`

org.osgi.framework

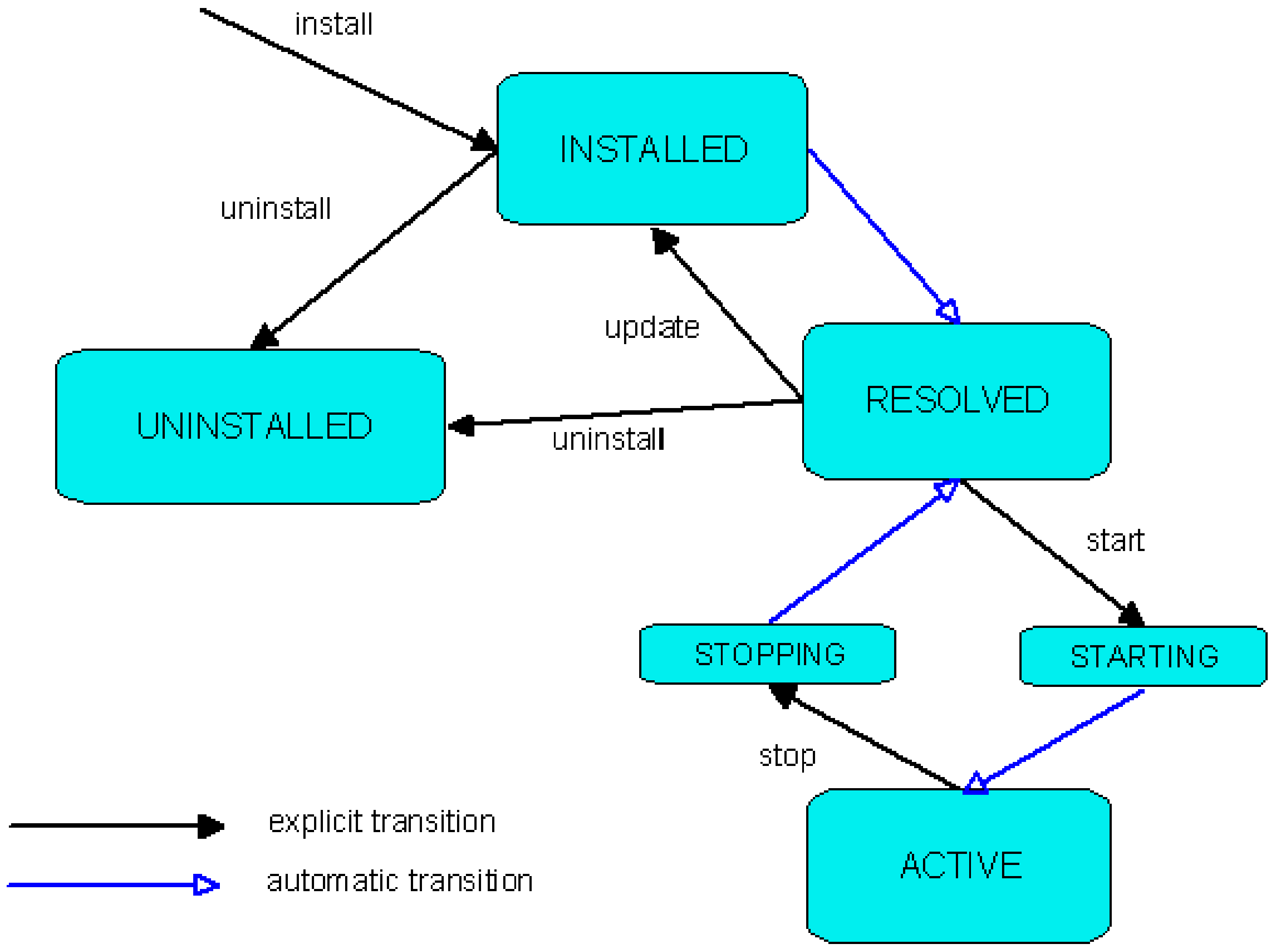
Interface Bundle

public interface **Bundle**

A bundle installed in a framework. The bundle is the access point to define the life cycle of the bundle. Each bundle installed in the framework will have an associated Bundle object.

A bundle will have a unique identity, a `long`, choosen 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 will create a new identity.

The Bundle has six states: `UNINSTALLED`, `INSTALLED`, `RESOLVED`, `STARTING`, `STOPPING`, and `ACTIVE`. The values assigned to these states have no specified ordering. They represent bit values that may be ORed together for the purposes of determining if a bundle is in one of a set of states.



A bundle should only be executing code when its state is in {`STARTING`, `ACTIVE`, `STOPPING`}. An `UNINSTALLED` bundle can never go back to another state. It is a zombie and can only be reached because invalid references are kept somewhere.

The framework is the only one that can create Bundle objects and these objects are only valid within the framework that created them.

Version:

1.0

Author:

Open Services Gateway Initiative

Field Summary

`static int`	**`ACTIVE`** Active state, the bundle is now running.
`static int`	**`INSTALLED`** Installed state, the bundle is installed but not yet resolved.
`static int`	**`RESOLVED`** Resolved state, the bundle is resolved and is able to be started.
`static int`	**`STARTING`** Starting state, the bundle is in the process of starting.
`static int`	**`STOPPING`** Stopping state, the bundle is in the process of stopping.
`static int`	**`UNINSTALLED`** Uninstalled state, the bundle is uninstalled and may not be used.

Method Summary

`long`	**`getBundleId`**`()` Retrieve the bundle's unique identifier, which the framework assigned to this bundle when it was installed.
`java.util.Dictionary`	**`getHeaders`**`()` Return the bundle's manifest headers and values from the manifest's preliminary section.
`java.lang.String`	**`getLocation`**`()` Retrieve the location identifier of the bundle.
`ServiceReference[]`	**`getRegisteredServices`**`()` Provides a list of `ServiceReference`s for the services registered by this bundle or `null` if the bundle has no registered services.

`ServiceReference[]`	**`getServicesInUse`**`()` Provides a list of `ServiceReference`s for the services this bundle is using, or `null` if the bundle is not using any services.
`int`	**`getState`**`()` Returns the current state of the bundle.
`boolean`	**`hasPermission`**`(java.lang.Object permission)` Determine whether the bundle has the requested permission.
`void`	**`start`**`()` Start this bundle.
`void`	**`stop`**`()` Stop this bundle.
`void`	**`uninstall`**`()` Uninstall this bundle.
`void`	**`update`**`()` Update this bundle.
`void`	**`update`**`(java.io.InputStream in)` Update this bundle from an InputStream.

Field Detail

UNINSTALLED

`public static final int` **`UNINSTALLED`**

Uninstalled state, the bundle is uninstalled and may not be used. The UNINSTALLED state is only be visible after a bundle is uninstalled. The bundle is in an unusable state and all references to the Bundle object should be released immediately.

The value of UNINSTALLED is 0x00000001.

INSTALLED

`public static final int` **`INSTALLED`**

Installed state, the bundle is installed but not yet resolved. The 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.

RESOLVED

`public static final int` **`RESOLVED`**

Resolved state, the bundle is resolved and is able to be started. The bundle is in the RESOLVED state when the framework has successfully resolved the bundle's code dependencies. These dependencies include:

- The bundle's class path from its `Bundle-ClassPath` manifest header.
- The bundle's native code from its `Bundle-NativeCode` manifest header.
- The bundle's package dependencies from its `Export-Package` and `Import-Package` manifest headers.

However, the bundle is not active yet. A bundle must be 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.

STARTING

`public static final int` **`STARTING`**

Starting state, the bundle is in the process of starting. The bundle is in the STARTING state when the `start()` method is active. The bundle will be in this state when the bundle's `BundleActivator.start` is called. If the `BundleActivator.start` method completes without exception, the bundle has successfully started and will move to the `ACTIVE` state.

The value of STARTING is 0x00000008.

STOPPING

`public static final int` **`STOPPING`**

Stopping state, the bundle is in the process of stopping. The bundle is in the STOPPING state when the `stop()` method is active. The bundle will be in this state when the bundle's `BundleActivator.stop` is called. When the `BundleActivator.stop` method completes the bundle is stopped and will move to the `RESOLVED` state.

The value of STOPPING is 0x00000010.

ACTIVE

`public static final int` **`ACTIVE`**

Active state, the bundle is now running. The bundle is in the ACTIVE state when it has been successfully started.

The value of ACTIVE is 0x00000020.

Method Detail

getState

```
public int getState()
```

Returns the current state of the bundle. A bundle can only be in one state at any time.

Returns:

element of {`UNINSTALLED`, `INSTALLED`, `RESOLVED`, `STARTING`, `STOPPING`, `ACTIVE`}

---

start

```
public void start()
          throws BundleException
```

Start this bundle.

The following steps are followed to start a bundle:

1. If the bundle is `UNINSTALLED` then an `IllegalStateException` is thrown.
2. If the bundle is `STARTING` or `STOPPING` then this method will wait for the bundle to change state before continuing. If this does not occur in a reasonable time, a `BundleException` is thrown to indicate the bundle was unable to be started.
3. If the bundle is `ACTIVE` then this method returns immediately.
4. If the bundle is not `RESOLVED`, an attempt is made to resolve the bundle. If the bundle cannot be resolved, a `BundleException` is thrown.
5. The state of the bundle is set to `STARTING`.
6. The `start` method of the bundle's `BundleActivator`, if one is specified, is called. If the `BundleActivator` is invalid or throws an exception, the state of the bundle is set back to `RESOLVED`, the bundle's listeners, if any, are removed, service's registered by the bundle, if any, are unregistered, and service's used by the bundle, if any, are released. A `BundleException` is then thrown.
7. It is recorded that this bundle has been started, so that when the framework is restarted, this bundle will be automatically started.
8. The state of the bundle is set to `ACTIVE`.
9. A `BundleEvent` of type `BundleEvent.STARTED` is broadcast.

Preconditons

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

Postconditons, 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 the 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.

java.lang.IllegalStateException - If the bundle has been uninstalled or the bundle tries to change its own state.

java.lang.SecurityException - If the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

---

stop

```
public void stop()
         throws BundleException
```

Stop this bundle. Any services registered by this bundle will be unregistered. Any services used by this bundle will be released. Any listeners registered by this bundle will be removed.

The following steps are followed to stop a bundle:

1. If the bundle is `UNINSTALLED` then an `IllegalStateException` is thrown.
2. If the bundle is `STARTING` or `STOPPING` then this method will wait for the bundle to change state before continuing. If this does not occur in a reasonable time, a `BundleException` is thrown to indicate the bundle was unable to be stopped.
3. If the bundle is not `ACTIVE` then this method returns immediately.
4. The state of the bundle is set to `STOPPING`.
5. It is recorded that this bundle has been stopped, so that when the framework is restarted, this bundle will not be automatically started.
6. The `stop` method of the bundle's `BundleActivator`, if one is specified, is called. If the `BundleActivator` throws an exception, this method will continue to stop the bundle. A `BundleException` will be thrown after completion of the remaining steps.
7. The bundle's listeners, if any, are removed, service's registered by the bundle, if any, are unregistered, and service's used by the bundle, if any, are released.
8. The state of the bundle is set to `RESOLVED`.
9. A `BundleEvent` of type `BundleEvent.STOPPED` is broadcast.

Preconditons

- getState() in {`ACTIVE`}.

Postconditons, 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 the bundle's BundleActivator could not be loaded or threw an exception.

java.lang.IllegalStateException - If the bundle has been uninstalled or the bundle tries to change its own state.

java.lang.SecurityException - If the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

---

update

```
public void update()
            throws BundleException
```

Update this bundle. If the bundle is `ACTIVE`, the bundle will be stopped before the update and started after the update successfully completes.

If the bundle that is being updated has exported any packages, it is the framework's responsibility to ensure that all bundles that are importing those packages (including the bundle that is exporting them) share the same version of the exported class files. In one implementation of the framework, updating a package may not have any effect on the importing bundles until the framework is restarted. Another framework implementation may choose to resolve all importing bundles against the updated class files, by possibly stopping and restarting them.

The following steps are followed to update a bundle:

1. If the bundle is `UNINSTALLED` then an `IllegalStateException` is thrown.
2. If the bundle is `ACTIVE` or `STARTING`, the bundle is stopped as described in the `stop()` method. If `stop()` throws an exception, the exception is rethrown terminating the update.
3. The location for the new version of the bundle is determined from either the manifest header `Bundle-UpdateLocation` if available or the original location.
4. The location is interpreted in an implementation dependent way (typically as a URL) and the new version of the bundle is obtained from the location.
5. The new version of the bundle is installed. If the framework is unable to install the new version of the bundle, the original version of the bundle will be restored and a `BundleException` will be thrown after completion of the remaining steps.
6. The state of the bundle is set to `INSTALLED`.
7. If the new version of the bundle was successfully installed, a `BundleEvent` of type `BundleEvent.UPDATED` is broadcast.
8. If the bundle was originally `ACTIVE`, the updated bundle is started as described in the `start()` method. If `start()` throws an exception, a `FrameworkEvent` of type

`FrameworkEvent.ERROR` is broadcast containing the exception.

Preconditions

- getState() not in {`UNINSTALLED`}.

Postconditons, no exceptions thrown

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

Postconditions, when an exception is thrown

- getState() in {`INSTALLED`,`RESOLVED`,`ACTIVE`}.
- Original bundle is still used, no update took place.

Throws:

BundleException - If the update fails.

java.lang.IllegalStateException - If the bundle has been uninstalled or the bundle tries to change its own state.

java.lang.SecurityException - If the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

See Also:

`stop()`, `start()`

---

update

```
public void update(java.io.InputStream in)
            throws BundleException
```

Update this bundle from an InputStream.

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

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

Parameters:

`in` - The InputStream from which to read the new bundle.

See Also:

`update()`

---

uninstall

```
public void uninstall()
               throws BundleException
```

Uninstall this bundle.

This causes the framework to notify other bundles that this bundle is being uninstalled, and then to put this bundle into the UNINSTALLED state. The framework will remove any resources related to this bundle that it can.

If this bundle has been exporting any packages, the framework may either:

- continue to make all packages exported by this bundle available to the importing bundles until the framework is relaunched (at which time, the importing bundles will be bound to another bundle exporting a compatible package, or they will not be started); or
- remove all packages exported by this bundle, possibly stopping all importing bundles and putting them into the INSTALLED state until another bundle offering a compatible package for export has been selected by the framework.

The following steps are followed to uninstall a bundle:

1. If the bundle is `UNINSTALLED` then an `IllegalStateException` is thrown.
2. If the bundle is `ACTIVE` or `STARTING`, the bundle is stopped as described in the `stop()` method. If `stop()` throws an exception, a `FrameworkEvent` of type `FrameworkEvent.ERROR` is broadcast containing the exception.
3. A `BundleEvent` of type `BundleEvent.UNINSTALLED` is broadcast.
4. The state of the bundle is set to `UNINSTALLED`.
5. The bundle and the persistent storage area provided for the bundle by the framework, if any, is removed.

Preconditions

- getState() not in {`UNINSTALLED`}.

Postconditons, no exceptions thrown

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

Postconditions, when an exception is thrown

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

Throws:

BundleException - If the uninstall failed.

java.lang.IllegalStateException - If the bundle has been uninstalled or the bundle tries to change its own state.

java.lang.SecurityException - If the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

See Also:

`stop()`

getHeaders

```
public java.util.Dictionary getHeaders()
```

Return the bundle's manifest headers and values from the manifest's preliminary section. That is all the manifest's headers and values prior to the first blank line.

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

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

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

This method will continue to return this information when the bundle is in the `UNINSTALLED` state.

Returns:

A `Dictionary` object containing the bundle's manifest headers and values.

Throws:

java.lang.SecurityException - If the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

getBundleId

```
public long getBundleId()
```

Retrieve the bundle's unique identifier, which the framework assigned to this bundle when it was installed.

The unique identifier has the following attributes:

- It is unique and persistent.
- The identifier is a long.
- Once its value is assigned to a bundle, that value is not reused for another bundle, even after the bundle is uninstalled.
- Its value does not change as long as the bundle remains installed.
- Its value does not change when the bundle is updated

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

Returns:

This bundle's unique identifier.

getLocation

`public java.lang.String **getLocation**()`

Retrieve the location identifier of the bundle. This is typically the location passed to `BundleContext.installBundle` when the bundle was installed. The location identifier of the bundle may change during bundle update. Calling this method while framework is updating the bundle results in undefined behavior.

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

Returns:

A string that is the location identifier of the bundle.

Throws:

java.lang.SecurityException - If the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

---

getRegisteredServices

`public ServiceReference[] **getRegisteredServices**()`

Provides a list of `ServiceReference`s for the services registered by this bundle or `null` if the bundle has no registered services.

If the Java runtime supports permissions, a `ServiceReference` to a service is included in the returned list if and 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, but the framework is a very dynamic environment and services can be modified or unregistered at anytime.

Returns:

An array of `ServiceReference` or `null`.

Throws:

java.lang.IllegalStateException - If the bundle has been uninstalled.

See Also:

`ServiceRegistration`, `ServiceReference`

---

getServicesInUse

`public ServiceReference[] **getServicesInUse**()`

Provides a list of `ServiceReference`s for the services this bundle is using, or `null` if the bundle is not using any services. A bundle is considered to be using a service if the bundle's use count for the service is greater than zero.

If the Java runtime supports permissions, a `ServiceReference` to a service is included in the returned list if and 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, but the framework is a very dynamic environment and services can be modified or unregistered at anytime.

Returns:

An array of `ServiceReference` or `null`.

Throws:

java.lang.IllegalStateException - If the bundle has been uninstalled.

See Also:

`ServiceReference`

---

hasPermission

public boolean **hasPermission**(java.lang.Object permission)

Determine whether the bundle has the requested permission.

If the Java runtime environment does not support permissions this method always returns `true`. The permission parameter 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.

Parameters:

`permission` - The requested permission.

Returns:

`true` if the bundle has the requested permission or the permissions possessed by the bundle imply the requested permission; `false` if the bundle does not have the permission or the permission parameter is not an `instanceof java.security.Permission`.

Throws:

java.lang.IllegalStateException - If the bundle has been uninstalled.

---

org.osgi.framework

Interface ServiceReference

---

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 (See `BundleContext.getService`). Every registered service has a unique ServiceRegistration object and may have multiple, distinct ServiceReference objects referring to it. ServiceReferences to the same ServiceRegistration instance are considered equal (i.e., their `equals()` method will return `true` when compared) and have the same `hashCode`. If the same service object is registered multiple times, ServiceReferences to different ServiceRegistrations are considered different.

Version:

1.0

Author:

Open Services Gateway Initiative

Method Summary

`Bundle`	**`getBundle`**`()` Return the bundle which registered the service.
`java.lang.Object`	**`getProperty`**`(java.lang.String key)` Get the value of a service's property.
`java.lang.String[]`	**`getPropertyKeys`**`()` Get the list of key names for the service's properties.

Method Detail

getProperty

`public java.lang.Object **getProperty**(java.lang.String key)`

Get the value of a service's property.

This method will continue to return property values after the service has been unregistered. This is so that references to unregistered service can be interrogated. (For example: ServiceReference objects stored in the log.)

Parameters:

`key` - Name of the property.

Returns:

Value of the property or `null` if there is no property by that name.

---

getPropertyKeys

`public java.lang.String[] **getPropertyKeys**()`

Get the list of key names for the service's properties.

This method will continue to return the keys after the service has been unregistered. This is so that references to unregistered service can be interrogated. (For example: ServiceReference objects stored in the log.)

Returns:

The list of property key names.

---

getBundle

`public Bundle **getBundle**()`

Return the bundle which registered the service.

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 which registered the service.

See Also:

`BundleContext.registerService(java.lang.String[], java.lang.Object, java.util.Dictionary)`

---

org.osgi.framework

Interface ServiceListener

---

public interface **ServiceListener**

extends java.util.EventListener

ServiceEvent listener. ServiceListener is an interface that may be implemented by a bundle programmer. A ServiceListener is registered with the framework using a `BundleContext.addServiceListener` method. ServiceListeners are called with a `ServiceEvent` when a service has been registered or modified, or when a service is in the process of unregistering.

`ServiceEvent` delivery to ServiceListeners are filtered by the filter specified when the listener was registered. If the Java runtime environment supports permissions, then additional filtering is done. `ServiceEvent`s are only delivered to the listener if the bundle which defines the listener object's class has the `ServicePermission` permission to `"get"` the service using at least one of the named classes the service was registered under.

Version:

1.0

Author:

Open Services Gateway Initiative

See Also:

`ServiceEvent`, `ServicePermission`

Method Summary

`void`	**`serviceChanged`**`(ServiceEvent event)` Receive notification that a service has had a change occur in it's lifecycle.

Method Detail

serviceChanged

```
public void serviceChanged(ServiceEvent event)
```

Receive notification that a service has had a change occur in it's lifecycle.

Parameters:

`event` - The ServiceEvent.

---

org.osgi.framework

Interface ServiceFactory

public interface **ServiceFactory**

Service factories allow services to provide customized service objects. In order to gain control over the specific service object given to a bundle using the service, a bundle programmer can register a `ServiceFactory` object instead of a service object when registering a service.

When this is done, the `BundleContext.getService(org.osgi.framework.ServiceReference)` method calls the service factory's `getService` to create a service object specifically for the requesting bundle. The service object returned by the service factory is cached by the framework until the bundle releases its use of the service.

When the bundle's use count for the service drops to zero (including the bundle stopping or the service being unregistered), the service factory's `ungetService` method is called.

ServiceFactory objects are only used by the framework and are not made available to other bundles.

Version:

1.0

Author:

Open Services Gateway Initiative

Method Summary

`java.lang.Object`	**`getService`**`(Bundle bundle, ServiceRegistration registration)` Create a service object.
`void`	**`ungetService`**`(Bundle bundle, ServiceRegistration registration, java.lang.Object service)` Release a service object.

Method Detail

getService

```
public java.lang.Object getService(Bundle bundle,
                                   ServiceRegistration registration)
```

Create a service object.

The framework invokes this method the first time a given bundle requests a service object using `BundleContext.getService(org.osgi.framework.ServiceReference)`. The factory can 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(org.osgi.framework.ServiceReference)` from the same bundle.

The framework will check the returned service object. If the service object is not an `instanceof` all the classes named when the service was registered, `null` is returned to the bundle.

Parameters:

`bundle` - The bundle using the service.

`registration` - The `ServiceRegistration` for the service.

Returns:

A service object that **must** be an `instanceof` all the classes named when the service was registered,

See Also:

`BundleContext.getService(org.osgi.framework.ServiceReference)`

---

ungetService

```
public void ungetService(Bundle bundle,
                         ServiceRegistration registration,
                         java.lang.Object service)
```

Release a service object.

The framework invokes this method when a service has been released by a bundle. The service object may be destroyed at this time.

Parameters:

`bundle` - The bundle releasing the service.

`registration` - The `ServiceRegistration` for the service.

`service` - The service object returned by a previous call to `getService`.

See Also:

`BundleContext.ungetService(org.osgi.framework.ServiceReference)`

---

org.osgi.framework

Interface FrameworkListener

public interface **FrameworkListener**
extends java.util.EventListener

FrameworkEvent listener. FrameworkListener is an interface that may be implemented by a bundle programmer. A FrameworkListener is registered with the framework using the `BundleContext.addFrameworkListener` method. FrameworkListeners are called with a `FrameworkEvent` when the framework starts and when asynchronous errors occur.

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
`FrameworkEvent`

Method Summary

`void`	`frameworkEvent(FrameworkEvent event)` Receive notification of a general framework event.

Method Detail

frameworkEvent

```
public void frameworkEvent(FrameworkEvent event)
```

Receive notification of a general framework event.

Parameters:
`event` - The FrameworkEvent.

---

org.osgi.framework

Interface Configurable

public interface **Configurable**

Interface implemented by services which support a configuration object. Configurable is an interface that may be implemented by a bundle programmer. The implementation of a service that is configurable should implement this interface. Bundles that wish to configure a service may test to see if the service object is an `instanceof Configurable`.

Version:
1.0

Author:
Open Services Gateway Initiative

Method Summary

`java.lang.Object`	`getConfigurationObject()` Retrieve the configuration object for a service.

Method Detail

getConfigurationObject

```
public java.lang.Object getConfigurationObject()
```

Retrieve the configuration object for a service.

Services implementing this interface should be careful when returning their configuration object since this object is probably sensitive. If the Java runtime environment supports permissions, it is recommended that the caller is checked for an appropriate permission before returning the configuration object. It is recommended that callers possessing the `AdminPermission` always be allowed to retrieve the configuration object.

Returns:
The configuration object for the service.

Throws:

java.lang.SecurityException - If the caller does not have an appropriate permission and the Java runtime environment supports permissions.

---

org.osgi.framework

Interface BundleListener

public interface **BundleListener**

extends java.util.EventListener

BundleEvent listener. BundleListener is an interface that may be implemented by a bundle programmer. A BundleListener is registered with the framework using the `BundleContext.addBundleListener` method. BundleListeners are called with a `BundleEvent` when a bundle has been installed, started, stopped, updated, or uninstalled.

Version:

1.0

Author:

Open Services Gateway Initiative

See Also:

`BundleEvent`

Method Summary

`void`	**`bundleChanged`**`(BundleEvent event)` Receive notification that a bundle has had a change occur in it's lifecycle.

Method Detail

bundleChanged

`public void` **`bundleChanged`**`(BundleEvent event)`

Receive notification that a bundle has had a change occur in it's lifecycle.

Parameters:

`event` - The BundleEvent.

org.osgi.framework

Class BundleEvent

```
java.lang.Object
  |
  +--java.util.EventObject
        |
        +--org.osgi.framework.BundleEvent
```

public class **BundleEvent**
extends java.util.EventObject

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

OSGi reserves the right to extend the set of types in the future.

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
Serialized Form

Field Summary

static int	**INSTALLED** A bundle has been installed.
static int	**STARTED** A bundle has been started.
static int	**STOPPED** A bundle has been stopped.
static int	**UNINSTALLED** A bundle has been uninstalled.
static int	**UPDATED** A bundle has been updated.

Fields inherited from class java.util.EventObject
`source`

Constructor Summary
`BundleEvent(int type, Bundle bundle)` Construct a bundle event.

Method Summary	
`Bundle`	`getBundle()` Retrieve the bundle who had a change occur in it's lifecycle.
`int`	`getType()` Retrieve the type of this event.

Methods inherited from class java.util.EventObject
`getSource, toString`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

INSTALLED

`public static final int` **`INSTALLED`**

A bundle has been installed.

The value of INSTALLED is 0x00000001.

See Also:

`BundleContext.installBundle(java.lang.String)`

STARTED

`public static final int` **`STARTED`**

A bundle has been started.

The value of STARTED is 0x00000002.

See Also:

`Bundle.start()`

STOPPED

`public static final int` **`STOPPED`**

A bundle has been stopped.

The value of STOPPED is 0x00000004.

See Also:

`Bundle.stop()`

UPDATED

`public static final int` **`UPDATED`**

A bundle has been updated.

The value of UPDATED is 0x00000008.

See Also:

`Bundle.update()`

UNINSTALLED

`public static final int` **`UNINSTALLED`**

A bundle has been uninstalled.

The value of UNINSTALLED is 0x00000010.

See Also:

`Bundle.uninstall()`

Constructor Detail

BundleEvent

```
public BundleEvent(int type,
                   Bundle bundle)
```

Construct a bundle event.

Parameters:

`type` - The event type.

`bundle` - The bundle who had a change occur in it's lifecycle.

Method Detail

getBundle

```
public Bundle getBundle()
```

Retrieve the bundle who had a change occur in it's lifecycle. This bundle is the source of the event.

Returns:

The bundle who had a change occur in it's lifecycle.

---

getType

```
public int getType()
```

Retrieve the type of this event. The type values are `INSTALLED`, `STARTED`, `STOPPED`, `UPDATED`, `UNINSTALLED`.

Returns:

The type of bundle lifecycle change.

---

org.osgi.framework

Class FrameworkEvent

```
java.lang.Object
  |
  +--java.util.EventObject
        |
        +--org.osgi.framework.FrameworkEvent
```

---

public class **FrameworkEvent**
extends java.util.EventObject

General framework event. The event class used when notifying listeners of general framework events. A type code is used to identify the event for future extendability.

OSGi reserves the right to extend the set of types in the future.

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
Serialized Form

Field Summary

static int	**ERROR** An error has occured.
static int	**STARTED** The framework has started.

Fields inherited from class java.util.EventObject
source

Constructor Summary

`FrameworkEvent(int type, Bundle bundle, java.lang.Throwable throwable)` Construct a framework event with a related bundle and exception.
`FrameworkEvent(int type, java.lang.Object source)` Construct a framework event.

Method Summary

`Bundle`	`getBundle()` Retrieve the bundle associated with the event.
`java.lang.Throwable`	`getThrowable()` Retrieve the exception associated with the event.
`int`	`getType()` Retrieve the type of this event.

Methods inherited from class java.util.EventObject
`getSource, toString`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

STARTED

`public static final int` **`STARTED`**

The framework has started. This event is broadcast when the framework has started after all installed bundle that are marked to be started have been started.

The value of STARTED is 0x00000001.

ERROR

```
public static final int ERROR
```

An error has occured. There was an error associated with a bundle.

The value of ERROR is 0x00000002.

Constructor Detail

FrameworkEvent

```
public FrameworkEvent(int type,
                      java.lang.Object source)
```

Construct a framework event. This constructor is used for framework events of type `STARTED`.

Parameters:

`type` - The event type.

`source` - The event source object. (This may not be `null`.)

FrameworkEvent

```
public FrameworkEvent(int type,
                      Bundle bundle,
                      java.lang.Throwable throwable)
```

Construct a framework event with a related bundle and exception. This constructor is used for framework events of type `ERROR`.

Parameters:

`type` - The event type.

`bundle` - The related bundle.

`throwable` - The related exception.

Method Detail

getThrowable

```
public java.lang.Throwable getThrowable()
```

Retrieve the exception associated with the event. If the event type is `ERROR`, this returns the exception related to the error.

Returns:

An exception if an `ERROR` event type or `null`.

---

getBundle

`public Bundle getBundle()`

Retrieve the bundle associated with the event. If the event type is `ERROR`, this returns the bundle related to the error. This bundle is also the source of the event.

Returns:

A bundle if an `ERROR` event type or `null`.

---

getType

`public int getType()`

Retrieve the type of this event. The type values are `STARTED`, `ERROR`.

Returns:

The type of bundle state change.

Overview **Package** **Class** **Tree** **Deprecated** **Index** **Help**

OSGi Service Gateway
Release 1.0 Final

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES**

SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

org.osgi.framework
Class PackagePermission

```
java.lang.Object
  |
  +--java.security.Permission
        |
        +--java.security.BasicPermission
              |
              +--org.osgi.framework.PackagePermission
```

---

public final class **PackagePermission**
extends java.security.BasicPermission

PackagePermission indicates a bundle's right to import or export a package. A package is a dot-separated string that defines a fully qualified Java package, e.g. `org.osgi.service.http`.

PackagePermission has two actions: "export" and "import". The export action also implies the import action.

Version:
: 1.0

Author:
: Open Services Gateway Initiative

See Also:
: Serialized Form

Field Summary

`static java.lang.String`	**`EXPORT`** The action string "export".
`static java.lang.String`	**`IMPORT`** The action string "import".

Constructor Summary

`PackagePermission(java.lang.String name, java.lang.String actions)` Define the permission to import and/or export a package.

Method Summary

`boolean`	`equals(java.lang.Object obj)` Checks two PackagePermission for equality.
`java.lang.String`	`getActions()` Return the canonical string representation of the actions.
`int`	`hashCode()` Returns the hash code value for this object.
`boolean`	`implies(java.security.Permission p)` Checks if the specified permission is "implied" by this object.

Methods inherited from class java.security.BasicPermission
`newPermissionCollection`

Methods inherited from class java.security.Permission
`checkGuard, getName, toString`

Methods inherited from class java.lang.Object
`clone, finalize, getClass, notify, notifyAll, wait, wait, wait`

Field Detail

EXPORT

```
public static final java.lang.String EXPORT
```

The action string "export".

IMPORT

```
public static final java.lang.String IMPORT
```

The action string "import".

Constructor Detail

PackagePermission

```
public PackagePermission(java.lang.String name,
                         java.lang.String actions)
```

Define the permission to import and/or export a package. The name is specified as a normal Java package name, with dots separating the parts. 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 wants to export a package must have the PackagePermission for "export" on that package. Similarly, a bundle that wants to import a package must have the PackagePermssion for "import" on that package.

Access permission is granted for both classes and resources.

Parameters:

`name` - Package name.

`actions` - "export", "import" (canonical order)

Method Detail

implies

```
public boolean implies(java.security.Permission p)
```

Checks 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 actions must either match or allow for the list for the target object to imply the target permission. 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
```

Overrides:

implies in class java.security.BasicPermission

Parameters:

`p` - the target permission to check.

Returns:

true if the specified permission is implied by this object, false if not.

---

getActions

`public java.lang.String **getActions**()`

Return the canonical string representation of the actions. Always returns present actions in the following order: export, import.

Overrides:

getActions in class java.security.BasicPermission

Returns:

The canonical string representation of the actions

---

equals

`public boolean **equals**(java.lang.Object obj)`

Checks two PackagePermission for equality. Checks that `obj` has the same package name and actions as this PackagePermission.

Overrides:

equals in class java.security.BasicPermission

Parameters:

`obj` - the object to test for equality.

Returns:

true if obj is a PackagePermission, and has the same package name and actions as this PackagePermission object. Otherwise, return false.

---

hashCode

```
public int hashCode()
```

Returns the hash code value for this object.

Overrides:

hashCode in class java.security.BasicPermission

Returns:

a hash code value for this object.

---

org.osgi.framework

Class ServiceEvent

```
java.lang.Object
  |
  +--java.util.EventObject
        |
        +--org.osgi.framework.ServiceEvent
```

---

public class **ServiceEvent**
extends java.util.EventObject

Service lifecycle change event. ServiceEvents are delivered to `ServiceListener`s when a change occurs in the service's lifecycle. A type code is used to identify the event for future extendability.

OSGi reserves the right to extend the set of types in the future.

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
Serialized Form

Field Summary

`static int`	**`MODIFIED`** The properties of a registered service have been modified.
`static int`	**`REGISTERED`** A service has been registered.
`static int`	**`UNREGISTERING`** A service is in the process of being unregistered.

Fields inherited from class java.util.EventObject
`source`

Constructor Summary

ServiceEvent(int type, ServiceReference reference) Construct a service event.

Method Summary

ServiceReference	**getServiceReference**() Retrieve a reference to the service who had a change occur in it's lifecycle.
int	**getType**() Retrieve the type of this event.

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

REGISTERED

```
public static final int REGISTERED
```

A service has been registered. This event is synchronously delivered AFTER the service has been registered.

The value of REGISTERED is 0x00000001.

See Also:

BundleContext.registerService(java.lang.String[], java.lang.Object, java.util.Dictionary)

MODIFIED

```
public static final int MODIFIED
```

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(java.util.Dictionary)`

UNREGISTERING

```
public static final int UNREGISTERING
```

A 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 unregistering the service.

The value of UNREGISTERING is 0x00000004.

See Also:

`ServiceRegistration.unregister()`, `BundleContext.ungetService(org.osgi.framework.ServiceReference)`

Constructor Detail

ServiceEvent

```
public ServiceEvent(int type,
                    ServiceReference reference)
```

Construct a service event.

Parameters:

`type` - The event type.

`reference` - A ServiceReference to the service who had a change occur in it's lifecycle.

Method Detail

getServiceReference

```
public ServiceReference getServiceReference()
```

Retrieve a reference to the service who had a change occur in it's lifecycle. This reference is the source of the event.

Returns:

A reference to the service who had a change occur in it's lifecycle.

---

getType

```
public int getType()
```

Retrieve the type of this event. The type values are `REGISTERED`, `MODIFIED`, `UNREGISTERING`.

Returns:

The type of service lifecycle change.

---

org.osgi.framework
Class ServicePermission

```
java.lang.Object
  |
  +--java.security.Permission
        |
        +--java.security.BasicPermission
              |
              +--org.osgi.framework.ServicePermission
```

public final class **ServicePermission**
extends java.security.BasicPermission

ServicePermission indicates a bundle's right to register or get a service.

The "register" permission will allow the bundle to register a service on the specified names. The "get" permission allows a bundle to see a service and get it. Permission to get a service is required to see events regarding the service. Untrusted bundles should not be able to detect the presence of certain services unless they have authority to get the specific service.

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
Serialized Form

Field Summary

static java.lang.String	**GET** The action string "get".
static java.lang.String	**REGISTER** The action string "register".

Constructor Summary

ServicePermission(java.lang.String name, java.lang.String actions) Create a new permission for Service.

Method Summary

boolean	**equals**(java.lang.Object obj) Checks two ServicePermission for equality.
java.lang.String	**getActions**() Return the canonical string representation of the actions.
int	**hashCode**() Returns the hash code value for this object.
boolean	**implies**(java.security.Permission p) Checks if this ServicePermission object "implies" the specified permission.

Methods inherited from class java.security.BasicPermission
newPermissionCollection

Methods inherited from class java.security.Permission
checkGuard, getName, toString

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

GET

```
public static final java.lang.String GET
```

The action string "get".

REGISTER

```
public static final java.lang.String REGISTER
```

The action string "register".

Constructor Detail

ServicePermission

```
public ServicePermission(java.lang.String name,
                         java.lang.String actions)
```

Create a new permission for Service. 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.

Parameters:

`name` - class name

`actions` - "get", "register" (canonical order)

Method Detail

implies

```
public boolean implies(java.security.Permission p)
```

Checks if this ServicePermission object "implies" the specified permission.

Overrides:

implies in class java.security.BasicPermission

Parameters:

`p` - the target permission to check.

Returns:

true if the specified permission is implied by this object, false if not.

---

getActions

`public java.lang.String` **`getActions`**`()`

Return the canonical string representation of the actions. Always returns present actions in the following order: get, register.

Overrides:

getActions in class java.security.BasicPermission

Returns:

The canonical string representation of the actions

---

equals

`public boolean` **`equals`**`(java.lang.Object obj)`

Checks two ServicePermission for equality. Checks that `obj` has the same class name and action as this ServicePermission.

Overrides:

equals in class java.security.BasicPermission

Parameters:

`obj` - the object to test for equality.

Returns:

true if obj is a ServicePermission, and has the same class name and actions as this ServicePermission object. Otherwise, return false.

---

hashCode

`public int` **`hashCode`**`()`

Returns the hash code value for this object.

Overrides:

hashCode in class java.security.BasicPermission

Returns:

a hash code value for this object.

---

org.osgi.framework

Interface BundleContext

public interface **BundleContext**

Bundle's execution context. Represents the execution context of a bundle within the framework. The context provides methods which allow the bundle to interact with the framework.

The provided methods allow a bundle to:

- Subscribe to the events published by the framework.
- Register services in the framework's service registry.
- Retrieve references to service from the framework's service registry.
- Get and release the service objects for a referenced service.
- Install new bundles into the framework.
- Get the list of installed bundles.
- Get the `Bundle` 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 the bundle when the bundle is started (`BundleActivator.start`). The same BundleContext object will be passed to the bundle when the bundle is stopped (`BundleActivator.stop`). The context is for the private use of the bundle and, in general, is not meant to be shared with other bundles. The context is used when accounting for services and event listeners.

The BundleContext object is only valid during an execution instance of the bundle. That is during the period from when the bundle is called at `BundleActivator.start` until after the bundle is called and returns from `BundleActivator.stop` (or `BundleActivator.start` terminates with an exception). If the context object is used after this, an `IllegalStateException` may be thrown. When the bundle is next restarted, a new BundleContext object will be created.

The framework is the only one that can create BundleContext objects and these objects are only valid within the framework that created them.

Note: A single virtual machine may host multiple framework instances at any given time, but objects created by one framework instance cannot be used by bundles running in the execution context of another framework instance.

Version:

1.0

Author:

Open Services Gateway Initiative

Method Summary

`void`	**`addBundleListener`**`(BundleListener listener)` Add a bundle listener.
`void`	**`addFrameworkListener`**`(FrameworkListener listener)` Add a general framework listener.
`void`	**`addServiceListener`**`(ServiceListener listener)` Add a service listener.
`void`	**`addServiceListener`**`(ServiceListener listener, java.lang.String filter)` Add a service listener with a filter.
`Bundle`	**`getBundle`**`()` Retrieve the Bundle object for the context bundle.
`Bundle`	**`getBundle`**`(long id)` Retrieve the bundle that has the given unique identifier.
`Bundle[]`	**`getBundles`**`()` Retrieve a list of all installed bundles.
`java.io.File`	**`getDataFile`**`(java.lang.String filename)` Creates a `File` object for a file in the persistent storage area provided for the bundle by the framework.
`java.lang.String`	**`getProperty`**`(java.lang.String key)` Retrieve the value of the named environment property.
`java.lang.Object`	**`getService`**`(ServiceReference reference)` Get a service's service object.
`ServiceReference`	**`getServiceReference`**`(java.lang.String clazz)` Get a service reference.
`ServiceReference[]`	**`getServiceReferences`**`(java.lang.String clazz, java.lang.String filter)` Get a list of service references.
`Bundle`	**`installBundle`**`(java.lang.String location)` Install a bundle from a location.
`Bundle`	**`installBundle`**`(java.lang.String location, java.io.InputStream in)` Install a bundle from an InputStream.
`ServiceRegistration`	**`registerService`**`(java.lang.String[] clazzes, java.lang.Object service, java.util.Dictionary properties)` Register a service with multiple names.

`ServiceRegistration`	`registerService(java.lang.String clazz, java.lang.Object service, java.util.Dictionary properties)` Register a service with a single name.
`void`	`removeBundleListener(BundleListener listener)` Remove a bundle listener.
`void`	`removeFrameworkListener(FrameworkListener listener)` Remove a framework listener.
`void`	`removeServiceListener(ServiceListener listener)` Remove a service listener.
`boolean`	`ungetService(ServiceReference reference)` Unget a service's service object.

Method Detail

getProperty

```
public java.lang.String getProperty(java.lang.String key)
```

Retrieve the value of the named environment property. Values are provided for the following properties:

`org.osgi.framework.version`
: The version of the framework.

`org.osgi.framework.vendor`
: The vendor of this framework implementation.

`org.osgi.framework.language`
: The language being used. See ISO 639 for possible values.

`org.osgi.framework.os.name`
: The name of the operating system of the hosting computer.

`org.osgi.framework.os.version`
: The version number of the operating system of the hosting computer.

`org.osgi.framework.processor`
: The name of the processor of the hosting computer.

Note: These last four properties are used by the `Bundle-NativeCode` manifest header's matching algorithm for selecting native code.

Parameters:

`key` - The name of the requested property.

Returns:

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

getBundle

```
public Bundle getBundle()
```

Retrieve the Bundle object for the context bundle.

The context bundle is defined as the bundle which is associated with this BundleContext. More specifically, the context bundle is defined to be the bundle which was given this BundleContext in it's `BundleActivator`.

Returns:

The context bundle's `Bundle` object.

Throws:

java.lang.IllegalStateException - If the context bundle has stopped.

---

installBundle

```
public Bundle installBundle(java.lang.String location)
                     throws BundleException
```

Install a bundle from a location. The bundle is obtained from the location parameter as interpreted by the framework in an implementation dependent way. Typically, location will most likely be a URL.

The following steps are followed to install a bundle:

1. If a bundle having the same location is already installed, the `Bundle` object for that bundle is returned.
2. The bundle's content is read from the location. If this fails, a `BundleException` is thrown.
3. The bundle's associated resources are allocated. The associated resources consist of at least a unique identifier and a persistent storage area, if the platform has file system support. If this step fails, a `BundleException` is thrown.
4. The state of the bundle is set to `INSTALLED`.
5. A `BundleEvent` of type `BundleEvent.INSTALLED` is broadcast.
6. The `Bundle` object for the newly installed bundle is returned.

Postconditons, 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.

Parameters:

`location` - The location identifier of the bundle to install.

Returns:

The `Bundle` object of the installed bundle.

Throws:

BundleException - If the install failed.

java.lang.SecurityException - If the caller does not have the `AdminPermission` and the Java runtime environment supports permissions.

---

installBundle

```
public Bundle installBundle(java.lang.String location,
                            java.io.InputStream in)
                     throws BundleException
```

Install a bundle from an InputStream.

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

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

Parameters:

`location` - The location identifier of the bundle to install.

`in` - The InputStream from which the bundle will be read.

Returns:

The `Bundle` of the installed bundle.

Throws:

BundleException - If the provided stream cannot be read.

See Also:

`installBundle(java.lang.String)`

---

getBundle

```
public Bundle getBundle(long id)
```

Retrieve the bundle that has the given unique identifier.

Parameters:

`id` - The identifier of the bundle to retrieve.

Returns:

A `Bundle` object, or `null` if the identifier doesn't match any installed bundle.

---

getBundles

```
public Bundle[] getBundles()
```

Retrieve a list of all installed bundles. The list is valid at the time of the call to getBundles, but the framework is a very dynamic environment and bundles can be installed or uninstalled at anytime.

Returns:

An array of `Bundle` objects, one object per installed bundle.

---

addServiceListener

```
public void addServiceListener(ServiceListener listener,
                               java.lang.String filter)
                        throws InvalidSyntaxException
```

Add a service listener with a filter. `ServiceListener`s are notified when a service has a lifecycle state change. See `getServiceReferences` for a description of the filter syntax. The listener is added to the context bundle's list of listeners. See `getBundle()` for a definition of context bundle.

The listener is called if the filter criteria is met. To filter based upon the class of the service, the filter should reference the "objectClass" property. If the filter parameter is `null`, all services are considered to match the filter.

If the Java runtime environment supports permissions, the service listener 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.

Parameters:

`listener` - The service listener to add.

`filter` - The filter criteria.

Throws:

InvalidSyntaxException - If the filter parameter contains an invalid filter string which cannot be parsed.

java.lang.IllegalStateException - If the context bundle has stopped.

See Also:

`ServiceEvent`, `ServiceListener`

---

addServiceListener

```
public void addServiceListener(ServiceListener listener)
```

Add a service listener.

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

See Also:

`addServiceListener(ServiceListener, String)`

---

removeServiceListener

`public void **removeServiceListener**(ServiceListener listener)`

Remove a service listener. The listener is removed from the context bundle's list of listeners. See `getBundle()` for a definition of context bundle.

If this method is called with a listener which is not registered, then this method does nothing.

Parameters:

`listener` - The service listener to remove.

Throws:

java.lang.IllegalStateException - If the context bundle has stopped.

---

addBundleListener

`public void **addBundleListener**(BundleListener listener)`

Add a bundle listener. `BundleListener`s are notified when a bundle has a lifecycle state change. The listener is added to the context bundle's list of listeners. See `getBundle()` for a definition of context bundle.

Parameters:

`listener` - The bundle listener to add.

Throws:

java.lang.IllegalStateException - If the context bundle has stopped.

See Also:

`BundleEvent`, `BundleListener`

---

removeBundleListener

`public void **removeBundleListener**(BundleListener listener)`

Remove a bundle listener. The listener is removed from the context bundle's list of listeners. See `getBundle()` for a definition of context bundle.

If this method is called with a listener which is not registered, then this method does nothing.

Parameters:

`listener` - The bundle listener to remove.

Throws:

java.lang.IllegalStateException - If the context bundle has stopped.

---

addFrameworkListener

`public void **addFrameworkListener**(FrameworkListener listener)`

Add a general framework listener. `FrameworkListener`s are notified of general framework events. The listener is added to the context bundle's list of listeners. See `getBundle()` for a definition of context bundle.

Parameters:

`listener` - The framework listener to add.

Throws:

java.lang.IllegalStateException - If the context bundle has stopped.

See Also:

`FrameworkEvent`, `FrameworkListener`

---

removeFrameworkListener

`public void **removeFrameworkListener**(FrameworkListener listener)`

Remove a framework listener. The listener is removed from the context bundle's list of listeners. See `getBundle()` for a definition of context bundle.

If this method is called with a listener which is not registered, then this method does nothing.

Parameters:

`listener` - The framework listener to remove.

Throws:

java.lang.IllegalStateException - If the context bundle has stopped.

---

registerService

```
public ServiceRegistration registerService(java.lang.String[] clazzes,
                                           java.lang.Object service,
                                           java.util.Dictionary properties)
```

Register a service with multiple names. This method registers the given service object with the given properties under the given class names. 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()` for a definition of context bundle. Other bundles can locate the service by using either the `getServiceReferences` or `getServiceReference` method.

A bundle can register a service object that implements the `ServiceFactory` interface to have more flexiblity in providing service objects to different bundles.

The following steps are followed to register a service:

1. If the service parameter is not a `ServiceFactory`, an `IllegalArgumentException` is thrown if the service parameter is not an `instanceof` all the classes named.
2. The service is added to the framework's service registry and may now be used by other bundles.
3. A `ServiceEvent` of type `ServiceEvent.REGISTERED` is synchronously sent.
4. A `ServiceRegistration` object for this registration is returned.

Parameters:

`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 "objectClass".

`service` - The service object or a `ServiceFactory` object.

`properties` - The properties for this service. The keys in the properties object must all be Strings. Changes should not be made to this object after calling this method. To update the service's properties call the `ServiceRegistration.setProperties` method. This parameter may be `null` if the service has no properties.

Returns:

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

Throws:

java.lang.IllegalArgumentException - If one of the following is true:

- The service parameter is null.
- The service parameter is not a `ServiceFactory` and is not an `instanceof` all the named classes in the clazzes parameter.

java.lang.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.

java.lang.IllegalStateException - If the context bundle has stopped.

See Also:

`ServiceRegistration`, `ServiceFactory`

---

registerService

```
public ServiceRegistration registerService(java.lang.String clazz,
                                           java.lang.Object service,
                                           java.util.Dictionary properties)
```

Register a service with a single name. This method registers the given service object with the given properties under the given class name.

This method is otherwise identical to `registerService(java.lang.String[], java.lang.Object, java.util.Dictionary)` and is provided as a convenience when the service parameter will only be registered under a single class name.

See Also:

`registerService(java.lang.String[], java.lang.Object, java.util.Dictionary)`

---

getServiceReferences

```
public ServiceReference[] getServiceReferences(java.lang.String clazz,
                                               java.lang.String filter)
                                        throws InvalidSyntaxException
```

Get a list of service references. Retrieves a list of `ServiceReference`s for services which implement and were registered under the named class and match the filter criteria.

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

The filter parameter is used to select registered service whose properties objects contain keys and values which satisfy the filter. The syntax of the filter parameter is the string representation of LDAP search filters as defined in RFC 1960: 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 API.

The string representation of an LDAP search filter is defined by the following grammar. It uses a prefix format.

```
   <filter> ::= '(' <filtercomp> ')'
   <filtercomp> ::= <and> | <or> | <not> | <item>
   <and> ::= '&' <filterlist>
   <or> ::= '|' <filterlist>
   <not> ::= '!' <filter>
   <filterlist> ::= <filter> | <filter> <filterlist>
   <item> ::= <simple> | <present> | <substring>
   <simple> ::= <attr> <filtertype> <value>
   <filtertype> ::= <equal> | <approx> | <greater> | <less>
   <equal> ::= '='
   <approx> ::= '~='
   <greater> ::= '>='
   <less> ::= '<='
   <present> ::= <attr> '=*'
   <substring> ::= <attr> '=' <initial> <any> <final>
   <initial> ::= NULL | <value>
   <any> ::= '*' <starval>
   <starval> ::= NULL | <value> '*' <starval>
   <final> ::= NULL | <value>
```

`<attr>` is a string representing an attributte, or key, in the properties objects of the registered services. Attribute names are not case sensitive; that is cn and CN both refer to the same attribute. `<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 '`(`' or '`)`', these characters should be escaped by preceding them with the backslash '`\`' character. Note that although both the `<substring>` and `<present>` productions can produce the `'attr=*'` construct, this construct is used only to denote a presence filter.

Examples of LDAP filters are:

```
   (cn=Babs Jensen)
   (!(cn=Tim Howes))
   (&(objectClass=Person)(|(sn=Jensen)(cn=Babs J*)))
   (o=univ*of*mich*)
```

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

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

The approximate match (`~=`) is implementation specific but should at least ignore case and white space differences. Optional are codes like soundex or other smart "closeness" comparisons.

Comparison of values is not straightforward. Strings are compared differently than numbers and it is possible for a key to have multiple values. Note that that keys in the properties object must always be strings. The comparison is defined by the object type of the key's value. The following rules apply for comparison:

Property Value Type	**Comparison Type**
String	String comparison
Integer, Long, Float, Double, Byte, Short, BigInteger, BigDecimal	numerical comparison
Character	character comparison
Boolean	equality comparisons only
[] (array)	recursively applied to values
Vector	recursively applied to elements

Note: arrays of primitives are also supported.

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

```
   Properties p = new Properties();
   p.put( "cn", new String[] { "a", "b", "c" } );
```

p will match `(cn=a)` and also `(cn=b)`

A filter with unrecognizable data types will be evaluated to be `false` .

The following steps are followed 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 named class. If the caller does not have the

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 the `clazz` parameter is not `null`, the set is further reduced to those services which are an `instanceof` and were registered under the named class.
4. An array of `ServiceReference` to the selected services is returned.

Parameters:

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

`filter` - The filter criteria.

Returns:

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

Throws:

InvalidSyntaxException - If the filter parameter contains an invalid filter string which cannot be parsed.

---

getServiceReference

```
public ServiceReference getServiceReference(java.lang.String clazz)
```

Get a service reference. Retrieves a `ServiceReference` for a service which implements the named class.

This reference is valid at the time of the call to this method, but since the framework is a very dynamic environment, services can be modified or unregistered at anytime.

This method is provided as a convenience for when the caller is interested in any service which implements a named class. This method is the same as calling `getServiceReferences` with a `null` filter string but only a single `ServiceReference` is returned.

Parameters:

`clazz` - The class name with which the service was registered.

Returns:

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

See Also:

`getServiceReferences(java.lang.String, java.lang.String)`

---

getService

`public java.lang.Object` **`getService`**`(ServiceReference reference)`

Get a service's service object. Retrieves the service object for a service. A bundle's use of a service is tracked by a use count. Each time a service's service object is returned by `getService(org.osgi.framework.ServiceReference)`, the context bundle's use count for the service is incremented by one. Each time the service is release by `ungetService(org.osgi.framework.ServiceReference)`, the context bundle's use count for the service is decremented by one. When a bundle's use count for a service drops to zero, the bundle should no longer use the service. See `getBundle()` 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 followed 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 now one and the service was registered with a `ServiceFactory`, the `ServiceFactory.getService` 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` is not an `instanceof` all the classes named when the service was registered or the `ServiceFactory` throws an exception, `null` is returned and a `FrameworkEvent` of type `FrameworkEvent.ERROR` is broadcast.
4. The service object for the service is returned.

Parameters:

`reference` - A reference to the service whose service object is desired.

Returns:

A service object for the service associated with this reference, or `null` if the service is not registered.

Throws:

java.lang.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.

java.lang.IllegalStateException - If the context bundle has stopped.

See Also:

`ungetService(org.osgi.framework.ServiceReference)`, `ServiceFactory`

ungetService

`public boolean **ungetService**(ServiceReference reference)`

Unget a service's service object. Releases the service object for a service. 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()` 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 followed 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 now zero and the service was registered with a `ServiceFactory`, the `ServiceFactory.ungetService` method is called to release the service object for the context bundle.
4. `true` is returned.

Parameters:

`reference` - A reference to the service to be released.

Returns:

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

Throws:

java.lang.IllegalStateException - If the context bundle has stopped.

See Also:

`getService(org.osgi.framework.ServiceReference)`, `ServiceFactory`

---

getDataFile

`public java.io.File **getDataFile**(java.lang.String filename)`

Creates a `File` object for a file in the persistent storage area provided for the bundle by the framework. If the platform does not have file system support, this method will return `null`.

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 the empty string ("") as the parameter. See `getBundle()` 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","execute","delete" for all files (recursively) in the persistent storage area provided for the context bundle by the framework.

Parameters:

`filename` - A relative name to the file to be accessed.

Returns:

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

Throws:

java.lang.IllegalStateException - If the context bundle has stopped.

Overview **Package** **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES**

SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

OSGi Service Gateway
Release 1.0 Final

---

org.osgi.framework

Class AdminPermission

```
java.lang.Object
  |
  +--java.security.Permission
        |
        +--java.security.BasicPermission
              |
              +--org.osgi.framework.AdminPermission
```

public final class **AdminPermission**
extends java.security.BasicPermission

The AdminPermission indicates the caller's right to perform life-cycle operations on or to get sensitive information about a bundle.

AdminPermission has no actions or target.

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

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
Serialized Form

Constructor Summary

`AdminPermission()` Creates a new AdminPermission object.
`AdminPermission(java.lang.String name, java.lang.String actions)` Creates a new AdminPermission object.

Method Summary

`boolean`	`equals(java.lang.Object obj)` Checks two AdminPermission objects for equality.
`boolean`	`implies(java.security.Permission p)` Checks if the specified permission is "implied" by this object.
`java.security.PermissionCollection`	`newPermissionCollection()` Returns a new PermissionCollection object for storing AdminPermission objects.

Methods inherited from class java.security.BasicPermission
`getActions, hashCode`

Methods inherited from class java.security.Permission
`checkGuard, getName, toString`

Methods inherited from class java.lang.Object
`clone, finalize, getClass, notify, notifyAll, wait, wait, wait`

Constructor Detail

AdminPermission

```
public AdminPermission()
```

Creates a new AdminPermission object. Its name is set to "AdminPermission".

AdminPermission

```
public AdminPermission(java.lang.String name,
                       java.lang.String actions)
```

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

Parameters:

`name` - Ignored; always set to "AdminPermission".

`actions` - Ignored.

Method Detail

implies

```
public boolean implies(java.security.Permission p)
```

Checks if the specified permission is "implied" by this object. This method returns `true` if the specified permission is an instance of `AdminPermission`, and `false` otherwise.

Overrides:

implies in class java.security.BasicPermission

Parameters:

`p` - the permission to check against.

Returns:

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

---

equals

```
public boolean equals(java.lang.Object obj)
```

Checks two AdminPermission objects for equality. Two AdminPermission objects are always equal.

Overrides:

equals in class java.security.BasicPermission

Parameters:

`obj` - the object we are testing for equality with this object.

Returns:

`true` if *obj* is an AdminPermission, `false` otherwise.

---

newPermissionCollection

```
public java.security.PermissionCollection newPermissionCollection()
```

Returns a new PermissionCollection object for storing AdminPermission objects.

Overrides:

newPermissionCollection in class java.security.BasicPermission

Returns:

a new PermissionCollection object suitable for storing AdminPermissions.

Package org.osgi.service.http

The OSGi HttpService Specification.

See:
Description

Interface Summary

HttpContext	HttpContext defines methods that HttpService may call to get information about a registration.
HttpService	HttpService allows other bundles in the OSGi Framework to dynamically register resources and servlets into the HttpService's URI namespace.

Exception Summary

NamespaceException	A NamespaceException is thrown to indicate an error with the caller's request to register a servlet or resources into HttpService's URI namespace.

Package org.osgi.service.http Description

The OSGi HttpService Specification.

HttpService allows other bundles in the OSGi Framework to register resources and servlets to be accessed via Hypertext Transfer Protocol (HTTP). HttpService may implement either HTTP/1.0 or HTTP/1.1.

Two entity types can be registered with HttpService: servlets and resources. A servlet is an object which implements the Java Servlet API. Registering a servlet gives that servlet control over some part of the URI namespace. Registering resources allows HTML files, GIF files, class files, etc. to be made visible in the URI namespace by the requesting bundle.

Registering Servlets

Servlets which are registered using the same HttpContext object will share the same ServletContext. This is, HttpService provides a one-to-one mapping between ServletContexts and HttpContexts. Servlets can be registered using the `registerServlet` method.

For example:

```
HttpContext context = new HttpContext() {
    public boolean handleSecurity(
        HttpServletRequest request,
        HttpServletResponse response) throws IOException {
        return(true);
    }
    public URL getResource(String name) {
        return(getClass().getResource(name));
    }
    public String getMimeType(String name) {
        return(null);
    }
};
Hashtable initparams = new Hashtable();
initparams.put("some-key-name", "some-value-string");
Servlet myServlet = new MyServlet();
httpService.registerServlet("/servletAlias",
                            myServlet,
                            initparams,
                            context);
/* myServlet has been registered and it's init method
   has been called */
...
httpService.unregister("/servletAlias");
/* myServlet has been unregistered and it's destroy method
   has been called */
```

This would register the servlet "myServlet" at alias "/servletAlias". A request for `http://myserver:port/servletAlias` would map to the servlet myServlet and it's `service` method will be called to process the request.

The context object in this example provides simple implementations of the HttpContext methods.

Registering Resources

Resources can be registered using the `registerResources` method.

For example:

```
HttpContext context = new HttpContext() {
    public boolean handleSecurity(
        HttpServletRequest request,
        HttpServletResponse response) throws IOException {
        return(true);
    }
    public URL getResource(String name) {
        return(getClass().getResource(name));
    }
    public String getMimeType(String name) {
        return(null);
    }
};
httpService.registerResources("/files",
                              "/bundlefiles",
                              context);
...
httpService.unregister("/files");
```

The example registers the resource name "/bundlefiles" to the alias "/files". A request for `http://myserver:port/files/myfile.htm` would map to the name "/bundlefiles/myfile.htm". The context object will be called at the `getResource` method to maps the resource name "/bundlefiles/myfile.htm" to a `URL`. HttpService will then use the URL to read the read the resource and respond to the request.

The context object in this example provides simple implementations of the HttpContext methods. The implementation of `getResource` uses the bundle's class loader to return an `URL` for the resource. More sophisticated implementations could filter the input name restricting the resources that may be returned or map the input name onto the file system.

Mapping HTTP Requests to Servlet and Resource Registrations

HttpService's URI namespace may be "shared". For example, caller A registers alias "/a" and caller B registers alias "/a/b". A request URI of "/a/b" or request URIs that start with "/a/b/" will be mapped to B's registration. Other request URIs that start with "/a" will not be mapped to B's registration and may be mapped to A's registration. This implies that one registration can "hide" part of another registration. Registrations for identical aliases are not allowed. If caller A registers "/myAlias" and then caller B tries to register "/myAlias", caller B will receive a NamespaceException and its resource or servlet will not be registered. However caller B can register "/myAlias/more" (if no other registration for this alias exists).

When an HTTP request comes in from a client, HttpService checks to see if the request URI matches any registered aliases. If it does, then we have a matching registration. If the registration corresponds to a servlet, then the servlet will be called at its `service` method to complete the HTTP request. If the registration corresponds to a resource, then a target resource name is constructed by substituting the alias name from the registration with the resource name from the registration. For example:

`registrationName+requestURI.substring(registrationAlias.length())`

The target resource name will be passed to the `getResource` method of the registration's `HttpContext` object. If the returned `URL` object is not `null`, then HttpService will return the contents of the `URL` to the client completing the HTTP request. If the returned `URL` object is `null`, then we continue as if there was no match.

If there is not a match, HttpService will attempt to match substrings of the requestURI to registered aliases. The substrings of the request URI are selected as follows: Remove the last '/' and everything to the right of it. HttpService will repeat this process until either a match is found or the substring is an empty string. If this happens, HttpService will return error Not Found(404) to the client.

For example, an HTTP request comes in with a request URI of "/a/b/foo.txt" and the only registered alias is "/a". The search for "/a/b/foo.txt" will not match an alias, therefore HttpService with search for aliases "/a/b" and then "/a". The search for alias "/a" will result in a match and its registration will be used.

---

org.osgi.service.http

Interface HttpContext

---

public interface **HttpContext**

HttpContext defines methods that HttpService may call to get information about a registration. Servlets and resources must be registered with a HttpContext object. Servlets which are registered using the same HttpContext object will share the same ServletContext.

This interface is implemented by users of HttpService.

Version:
1.0

Author:
Open Services Gateway Initiative

Method Summary

`java.lang.String`	`getMimeType(java.lang.String name)` Map a name to a MIME type.
`java.net.URL`	`getResource(java.lang.String name)` Map a resource name to a URL.
`boolean`	`handleSecurity(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)` Handle security for a request.

Method Detail

handleSecurity

```
public boolean handleSecurity(javax.servlet.http.HttpServletRequest request,
                              javax.servlet.http.HttpServletResponse response)
                       throws java.io.IOException
```

Handle security for a request. HttpService calls this method prior to servicing a request. This method controls whether a request is processed in the normal manner or an error is returned.

If a request requires authentication and the `Authorization` header in the request is missing or not acceptable, then this method should set the `WWW-Authenticate` header in the response object, set the status in the response object to Unauthorized(401) and return `false`. See also HTTP Authentication: Basic and Digest Access Authentication.

If a request requires a secure connection and the `getScheme` method in the request does not return `https` or some other acceptable secure protocol, then this method should set the status in the response object to Forbidden(403) and return `false`.

When this method returns `false`, HttpService will send the response back to the client completing the request. When this method returns `true`, HttpService will proceed with servicing the request.

Parameters:

`request` - the HTTP request

`response` - the HTTP response

Returns:

`true` if the request should be serviced, `false` if the request should not be serviced and HttpService will send the response back to the client.

Throws:

java.io.IOException - may be thrown by this method. If this occurs HttpService will terminate the request and close the socket.

---

getResource

`public java.net.URL **getResource**(java.lang.String name)`

Map a resource name to a URL. Called by HttpService to map a resource name to a URL. For servlet registrations, HttpService will call this method to support the `ServletContext` methods `getResource` and `getResourceAsStream`. For resource registrations, HttpService will call this method to locate the named resource. The context can control from where resources come. For example, the resource can be mapped to a file in the bundle's persistent storage area via

```
 bundleContext.getDataFile(name).toURL()
```

or to a resource in the context's bundle via

```
 this.getClass().getResource(name)
```

Parameters:

`name` - the name of the requested resource

Returns:

URL that HttpService can use to read the resource or `null` if the resource does not exist.

---

getMimeType

`public java.lang.String **getMimeType**(java.lang.String name)`

Map a name to a MIME type. Called by HttpService to determine the MIME type for the name. For servlet registrations, HttpService will call this method to support the `ServletContext` method `getMimeType`. For resource registrations, HttpService will call this method to determine the MIME type for the `Content-Type` header in the response.

Parameters:

`name` - determine the MIME type for this name.

Returns:

MIME type (e.g. `text/html`) of the name or `null` to indicate that HttpService should determine the MIME type.

---

org.osgi.service.http

Interface HttpService

public interface **HttpService**

HttpService allows other bundles in the OSGi Framework to dynamically register resources and servlets into the HttpService's URI namespace. A Bundle may later unregister its resources or servlets.

Version:
1.0

Author:
Open Services Gateway Initiative

Method Summary

`void`	`registerResources(java.lang.String alias, java.lang.String name, HttpContext context)` Register resources into the URI namespace.
`void`	`registerServlet(java.lang.String alias, javax.servlet.Servlet servlet, java.util.Dictionary initparams, HttpContext context)` Register a servlet into the URI namespace.
`void`	`unregister(java.lang.String alias)` Unregisters a previous registration done by registerServlet or registerResources.

Method Detail

registerServlet

```
public void registerServlet(java.lang.String alias,
                            javax.servlet.Servlet servlet,
                            java.util.Dictionary initparams,
                            HttpContext context)
                     throws javax.servlet.ServletException,
                                              NamespaceException
```

Register a servlet into the URI namespace. The alias is the name in the URI namespace of HttpService at which the registration will be mapped. An alias must begin with slash ('/') and must not end with slash ('/'). See also Mapping HTTP Requests to Servlet and Resource Registrations.

HttpService will call the `init` method of the servlet before returning.

```
 httpService.registerServlet("/myservlet",
                             servlet,
                             initparams,
                             context);
```

Servlets which are registered with the same HttpContext object will share the same ServletContext. HttpService will call the HttpContext parameter to support the `ServletContext` methods `getResource`, `getResourceAsStream` and `getMimeType` and to handle security for requests.

Parameters:

`alias` - name in the URI namespace at which the servlet is registered

`servlet` - the servlet object to register

`initparams` - initialization parameters for the servlet or `null` if there are none. This parameter is used by the servlet's `ServletConfig` object.

`context` - the HttpContext object for the registered servlet

Throws:

NamespaceException - if the register fails because the alias is already in use.

javax.servlet.ServletException - if the servlet's `init` method throws an exception

java.lang.IllegalArgumentException - if any of the parameters are invalid

---

registerResources

```
public void registerResources(java.lang.String alias,
                              java.lang.String name,
                              HttpContext context)
                       throws NamespaceException
```

Register resources into the URI namespace. The alias is the name in the URI namespace of HttpService at which the registration will be mapped. An alias must begin with slash ('/') and must not end with slash ('/'). The name parameter must also not end with slash ('/'). See also Mapping HTTP Requests to Servlet and Resource Registrations.

For example, suppose the resource name /tmp is registered to the alias /files. A request for /files/foo.txt will map to the resource name /tmp/foo.txt.

```
 httpservice.registerResources("/files",
                                  "/tmp",
                                  context);
```

HttpService will call the HttpContext parameter to map resource names to URLs and MIME types and to handle security for requests.

Parameters:

`alias` - name in the URI namespace at which the resources are registered

`name` - the base name of the resources that will be registered

`context` - the HttpContext object for the registered resources

Throws:

NamespaceException - if the registration fails because the alias is already in use.

java.lang.IllegalArgumentException - if any of the parameters are invalid

unregister

```
public void unregister(java.lang.String alias)
```

Unregisters a previous registration done by registerServlet or registerResources. After this call, the registered alias in the URI namespace will no longer be available. If the registration was for a servlet, HttpService will call the `destroy` method of the servlet before returning.

If the bundle which performed the registration is stopped or otherwise "unget"s HttpService without calling `unregister`, then HttpService will automatically unregister the registration. However, if the registration was for a servlet, the `destroy` method of the servlet will not be called in this case since the bundle may be stopped. `unregister` must be explicitly called to cause the `destroy` method of the servlet to be called. This can be done in the `BundleActivator.stop` method of the bundle registering the servlet.

Parameters:

`alias` - name in the URI namespace of the registration to unregister

Throws:

java.lang.IllegalArgumentException - if there is no registration for the alias or the calling bundle was not the bundle which registered the alias.

org.osgi.service.http

Class NamespaceException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--org.osgi.service.http.NamespaceException
```

public class **NamespaceException**
extends java.lang.Exception

A NamespaceException is thrown to indicate an error with the caller's request to register a servlet or resources into HttpService's URI namespace. This exception indicates that the requested alias is already registered.

Version:
1.0

Author:
Open Services Gateway Initiative

See Also:
Serialized Form

Constructor Summary

`NamespaceException(java.lang.String message)` Construct a NamespaceException with a detail message.
`NamespaceException(java.lang.String message, java.lang.Throwable exception)` Construct a NamespaceException with a detail message and a nested exception.

Method Summary

`java.lang.Throwable`	`getException()` Returns the nested exception.

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

NamespaceException

```
public NamespaceException(java.lang.String message)
```

Construct a NamespaceException with a detail message.

Parameters:

`message` - the detail message

NamespaceException

```
public NamespaceException(java.lang.String message,
                          java.lang.Throwable exception)
```

Construct a NamespaceException with a detail message and a nested exception.

Parameters:

`message` - the detail message

`exception` - the nested exception

Method Detail

getException

`public java.lang.Throwable getException()`

Returns the nested exception.

Returns:

the nested exception or `null` if there is no nested exception.

Package org.osgi.service.log

The OSGi LogService Specification.

See:
Description

Interface Summary	
LogEntry	The LogEntry interface provides the methods to access the information contained in an individual LogService log entry.
LogListener	The LogListener interface is used to subscribe to LogEntry objects from the LogReaderService.
LogReaderService	The LogReaderService provides methods to read LogEntry objects from the log.
LogService	The LogService provides methods for bundles to write messages to the log.

Package org.osgi.service.log Description

The OSGi LogService Specification.

The LogService takes log requests from bundles and the LogReaderService allows other bundles to read entries from the log. Although you can ask the LogService to log any message, it is primarily intended for reporting events and error conditions.

In general, the LogService interface provides the means for:

- Specifying the message and/or exception to be logged.
- Supplying a log level signifying severity of the message being logged.
- Specifying the service associated with the log requests.

The LogReaderService interface provides the means for:

- Getting recent past log entries.
- Getting notified of new log entries.

LogService

Making Log Requests

Callers make log requests through LogService after they get the Service object from the OSGi Framework. In the following example, the caller writes a message into the log:

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

`myServiceReference` identifies the originator of the log request. The provided level `LogService.LOG_INFO` indicates that this is informational.

Following is another example that records error conditions:

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

Note that in addition to the error message, the exception itself is also logged .

Methods for Logging

The LogService provides for the logging of different types of messages:

- Log a simple message at the given log level.
- Log a message with an exception at the given log level.

Although it is possible to call log methods without providing a service description, it is recommended that the caller supplies this parameter whenever appropriate.

Log Level and Error Severity

The LogService expects a level indicating error severity. This can be used to filter log messages according to the level of severity. The LogService interface defines the various severity levels.

Callers must supply log levels that they deem appropriate when making log requests.

Event Listening

The LogService also serves as an event listener for the OSGi Framework. There is no dedicated error handling service in the Framework. Because events are broadcast in the Framework, other interested listeners can still have the opportunity to receive the events. The LogService will log all Framework events at the `LogService.LOG_INFO` level, except for FrameworkErrorEvents which will be logged at the `LogService.LOG_ERROR` level.

LogReaderService

In addition to producing log entries a bundle programmer might also be interested in receiving log entries either as they happen or entries that have already occurred. In secure implementations of the Framework, a bundle wishing to use the LogReaderService must have the appropriate permission.

Retrieving past log entries

Log entries from the past are obtained via the `getLog` method. This method returns an enumeration of LogEntries. The LogEntries will be ordered with the most recent entry first. It should be noted that the size of the log is implementation specific. Also, how far into the past the log goes will depend upon the size of the log. Finally, not all log entries will be recorded in the past log. Debug log entries, in particular, may not be recorded.

Subscribing to the LogReaderService

A bundle that is interested in the log entries will probably want to process log messages as they happen. Unlike the past log, all logging messages will be sent to a subscriber of the LogReaderService. A subscriber to the LogReaderService must implement the LogListener interface. The subscriber then subscribes to the LogReaderService using the `addLogListener` method. After starting the subscription, each time a message is logged, the `logged` method of the subscriber's LogListener will be called with a LogEntry object for the message that was logged.

---

org.osgi.service.log
Interface LogEntry

---

public interface **LogEntry**

The LogEntry interface provides the methods to access the information contained in an individual LogService log entry. A LogEntry may be acquired from the `LogReaderService.getLog` method or by registering a `LogListener`.

Version:
1.0

Author:
Open Services Gateway Initiative

Method Summary

`Bundle`	**`getBundle`**`()` The bundle that created the LogEntry.
`java.lang.Throwable`	**`getException`**`()` The exception object associated with the LogEntry.
`int`	**`getLevel`**`()` The severity level of the LogEntry.
`java.lang.String`	**`getMessage`**`()` The human readable message associated with the LogEntry.
`ServiceReference`	**`getServiceReference`**`()` The ServiceReference for the service associated with the LogEntry.
`long`	**`getTime`**`()` The value of `System.currentTimeMillis()` at the time the LogEntry was created.

Method Detail

getBundle

`public Bundle getBundle()`

The bundle that created the LogEntry.

Returns:

The bundle that created the LogEntry or `null` if no bundle is associated with the LogEntry.

---

getServiceReference

`public ServiceReference getServiceReference()`

The ServiceReference for the service associated with the LogEntry.

Returns:

`ServiceReference` for the service associated with the LogEntry or `null` if no `ServiceReference` was provided.

---

getLevel

`public int getLevel()`

The severity level of the LogEntry. This is one of the severity levels defined by `LogService`.

Returns:

Severity level of the LogEntry.

See Also:

`LogService.LOG_ERROR`, `LogService.LOG_WARNING`, `LogService.LOG_INFO`, `LogService.LOG_DEBUG`

---

getMessage

`public java.lang.String getMessage()`

The human readable message associated with the LogEntry.

Returns:

`String` containing the message associated with the LogEntry.

---

getException

```
public java.lang.Throwable getException()
```

The exception object associated with the LogEntry.

Returns:

`Throwable` object of the exception associated with the LogEntry or `null` if no exception is associated.

---

getTime

```
public long getTime()
```

The value of `System.currentTimeMillis()` at the time the LogEntry was created.

Returns:

The system time in milliseconds when the LogEntry was created

---

---

org.osgi.service.log

Interface LogListener

public interface **LogListener**

extends java.util.EventListener

The LogListener interface is used to subscribe to LogEntry objects from the LogReaderService. A LogListener object may be registered with the `LogReaderService` using the `addLogListener` method. After the listener is registered, it will be called at the `log` method for each `LogEntry` created. The listener may be unregistered by calling the `removeLogListener` method.

Version:
1.0

Author:
Open Services Gateway Initiative

Method Summary

`void`	`logged(LogEntry entry)` Listener method called for each LogEntry created.

Method Detail

logged

```
public void logged(LogEntry entry)
```

Listener method called for each LogEntry created. As with all event listeners, this method should return to its caller as soon as possible.

Parameters:
`entry` - A `LogEntry` object containing log information.

See Also:
`LogEntry`

---

org.osgi.service.log

Interface LogReaderService

---

public interface **LogReaderService**

The LogReaderService provides methods to read LogEntry objects from the log. Two ways are provided to read `LogEntry` objects.

The primary way to receive LogEntry objects is to register a `LogListener` object which will be called at the `logged` method for each entry added to the log.

To receive past LogEntry objects, `getLog` can be called which will return an Enumeration of all LogEntry objects in the log.

Version:
1.0

Author:
Open Services Gateway Initiative

Method Summary

`void`	`**addLogListener**(LogListener listener)` Subscribe to LogEntry objects.
`java.util.Enumeration`	`**getLog**()` Returns an enumeration of all LogEntry objects in the log.
`void`	`**removeLogListener**(LogListener listener)` Unsubscribe to LogEntry objects.

Method Detail

addLogListener

```
public void addLogListener(LogListener listener)
```

Subscribe to LogEntry objects. Registers a LogListener object with the LogReaderService. This object will be called at the `logged` method for each LogEntry placed into the log.

When a bundle which registers a LogListener is stopped or otherwise releases the LogReaderService, the LogReaderService must remove all the bundle's listeners.

Parameters:

`listener` - A `LogListener` object to register to receive `LogEntry` object's.

---

removeLogListener

```
public void removeLogListener(LogListener listener)
```

Unsubscribe to LogEntry objects. Unregisters a LogListener object from the LogReaderService.

Parameters:

`listener` - A `LogListener` object to unregister.

---

getLog

```
public java.util.Enumeration getLog()
```

Returns an enumeration of all LogEntry objects in the log. Each element of the enumeration is a LogEntry object. The LogEntries will be ordered with the most recent entry first. Whether the enumeration is of all LogEntry objects since the LogService was started or some recent past is implementation specific. It is also implementation specific as to whether informational and debug LogEntry objects are included in the enumeration.

---

org.osgi.service.log
Interface LogService

public interface **LogService**

The LogService provides methods for bundles to write messages to the log. Methods are provide to log messages with or without a ServiceDescription or an exception. Messages must be logged with a log level. The log levels have the following hierarchy:

1. `LOG_ERROR`
2. `LOG_WARNING`
3. `LOG_INFO`
4. `LOG_DEBUG`

Version:
1.0

Author:
Open Services Gateway Initiative

Field Summary

`static int`	**`LOG_DEBUG`** A debugging message.
`static int`	**`LOG_ERROR`** An error message.
`static int`	**`LOG_INFO`** An informational message.
`static int`	**`LOG_WARNING`** A warning message.

Method Summary

`void`	**`log`**`(int level, java.lang.String message)` Log a message.

void	**log**(int level, java.lang.String message, java.lang.Throwable exception) Log a message with an exception.
void	**log**(ServiceReference sr, int level, java.lang.String message) Log a message associated with a specific Service.
void	**log**(ServiceReference sr, int level, java.lang.String message, java.lang.Throwable exception) Log a message with an exception associated with a specific Service.

Field Detail

LOG_ERROR

public static final int **LOG_ERROR**

An error message. The bundle or service may not be functional.

LOG_WARNING

public static final int **LOG_WARNING**

A warning message. The bundle or service is still functioning but may experience problems in the future because of the condition.

LOG_INFO

public static final int **LOG_INFO**

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

LOG_DEBUG

public static final int **LOG_DEBUG**

A debugging message. Used for problem determination and may be meaningless to anyone but the developer.

Method Detail

log

```
public void log(int level,
                java.lang.String message)
```

Log a message. The ServiceDescription field and the Throwable field of the LogEntry will be set to null.

Parameters:

`level` - The severity of the message. (Should be one of the four predefined severities.)

`message` - Human readable string describing the condition.

---

log

```
public void log(int level,
                java.lang.String message,
                java.lang.Throwable exception)
```

Log a message with an exception. The ServiceDescription field of the LogEntry will be set to null.

Parameters:

`level` - The severity of the message. (Should be one of the four predefined severities.)

`message` - Human readable string describing the condition.

`exception` - The exception that reflects the condition.

---

log

```
public void log(ServiceReference sr,
                int level,
                java.lang.String message)
```

Log a message associated with a specific Service. The Throwable field of the LogEntry will be set to null.

Parameters:

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

`level` - The severity of the message. (Should be one of the four predefined severities.)

`message` - Human readable string describing the condition.

---

log

```
public void log(ServiceReference sr,
                int level,
                java.lang.String message,
                java.lang.Throwable exception)
```

Log a message with an exception associated with a specific Service.

Parameters:

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

`level` - The severity of the message. (Should be one of the four predefined severities.)

`message` - Human readable string describing the condition.

`exception` - The exception that reflects the condition.

Package org.osgi.service.device

The OSGi Device Access Specification.

See:
Description

Interface Summary	
Device	The Device interface should be implemented by services wishing to be discovered by the device manager.
Driver	A `Driver` service should be registered by each driver wishing to attach to device services provided by other drivers.
DriverLocator	A `DriverLocator` can find and load device driver bundles given a property set.

Package org.osgi.service.device Description

The OSGi Device Access Specification.

The Device Manager service listens for new Device services and attaches drivers on top of devices.

Device manager

The device manager registers as service listener in the framework, detecting all newly installed Device services. For each new Device service, a list of driver bundles are located and installed. Then, all Driver services are located from framework and tested in a *bidding phase*, where each Driver may inspect the Device service. The highest bidding Driver gets *attached* to the device.

Device Manager Algorithm

- **Device detection**
  Listen for all new Device Services implementing `org.osgi.service.device.Device`

- For each new Device service:
- **Location phase**
  Use DriverLocators to locate additional drivers
  - Get all DriverLocator services
  - Query each DriverLocator for driver IDs by calling `locator.findDriver(Device)`
  - Check which drivers are already present in Framework
  - Load, install and start new driver bundles by calling `locator.loadDriver(driver ID)` and `Framework.install(stream, driver id)`
- **Bidding phase**
  Let each driver bid on the device.
  - Get all registered Driver services (implementing org.osgi.service.device.Driver) from Framework.
  - For each Driver service:
    - Call the Driver service' `match()` method with the Device as argument.
    - Select the highest bidding Driver service
- **Attach phase**
  Attach highest bidder to device.
  - Call the Driver service's `attach()` method.
  - If `attach()` returns null, do nothing more.
  - If `attach()` returns a new driver ID, repeat the location phase, but exclude the referring driver from the next match round.
- **Cleanup phase**
  Uninstall idle drivers.

Driver bundle ID

Each driver bundle is expected to have a universally unique string ID. This ID is used by the device manager to resolve driver revisions and is also used as framework location ID.

DriverLocator service

The Device Manager uses zero or more DriverLocator services to find new driver bundles to install. DriverLocators are typically implementation specific for a specific type of driver databases and networ setups.

See Also:

`DriverLocator`

org.osgi.service.device

Interface Device

public interface **Device**

The Device interface should be implemented by services wishing to be discovered by the device manager. Concrete devices subclass this interface adding methods appropriate to the device category.

If no drivers are interested in this device, the `noDriverFound()` method is called.

See Also:

`Device`

Field Summary

`static int`	**`MATCH_NONE`** Return value from `Driver.match()` if the driver does not match the device.

Method Summary

`void`	**`noDriverFound`**`()` Called by the device manager after it has failed to attach any driver to the device.

Field Detail

MATCH_NONE

```
public static final int MATCH_NONE
```

Return value from `Driver.match()` if the driver does not match the device.

Method Detail

noDriverFound

```
public void noDriverFound()
```

Called by the device manager after it has failed to attach any driver to the device.

If the device can be configured in alternate ways, the driver may respond by unregistering the device service and registering a different device service instead.

org.osgi.service.device

Interface Driver

public interface **Driver**

A `Driver` service should be registered by each driver wishing to attach to device services provided by other drivers. For each newly discovered `Device`, the device manager enters a bidding phase. The `Driver` whose `match()` method bids the highest for a particular device will be instructed by the device manager to attach to the device.

See Also:

`Device`

Method Summary

`java.lang.String`	`attach(ServiceReference reference)` Attach this driver to the device represented by the given ServiceReference.
`int`	`match(ServiceReference reference)` Check whether this driver can be attached to the device represented by the given ServiceReference, and return a value indicating how well this driver can support the given device, or `Device.MATCH_NONE` if it cannot support the given device at all.

Method Detail

match

```
public int match(ServiceReference reference)
          throws java.lang.Exception
```

Check whether this driver can be attached to the device represented by the given ServiceReference, and return a value indicating how well this driver can support the given device, or `Device.MATCH_NONE` if it cannot support the given device at all.

The value returned must be one of the possible match values defined in the Device subinterface corresponding to the given device.

In order to make its decision, this driver may simply examine the properties associated with the given device, or may get the referenced service object (representing the actual physical device) to talk to it, as long as it ungets the service and returns the physical device to a normal state before this method returns.

A driver should always return the same match code whenever it is presented with the same device.

The match function is called by the device manager during the matching process.

Parameters:

`reference` - the `ServiceReference` of the device to match

Returns:

value indicating how well this driver can support the given device, or Device.MATCH_NONE if it cannot support the device at all

Throws:

java.lang.Exception - the driver cannot examine the device

---

attach

```
public java.lang.String attach(ServiceReference reference)
                        throws java.lang.Exception
```

Attach this driver to the device represented by the given ServiceReference.

A return value of `null` indicates that this driver has successfully attached to the given device. If this driver is unable to attach to the given device, but knows of a more suitable driver, it must return the ID of that driver. This allows for the implementation of referring drivers whose only purpose is to refer to other drivers capable of handling a given device.

After having attached to the device, this driver is expected to register the device as a new service exposing driver-specific functionality.

This method is called by the device manager.

Parameters:

`reference` - the ServiceReference of the device to attach to

Returns:

`null` if this driver has successfully attached to the given device, or the ID of a more suitable driver

Throws:

java.lang.Exception - the driver cannot attach to the given device and does not know of a more suitable driver

---

org.osgi.service.device

Interface DriverLocator

public interface **DriverLocator**

A `DriverLocator` can find and load device driver bundles given a property set. Each driver is represented by a unique ID.

DriverLocator services provide the mechanism for dynamically downloading new device driver bundles into an OSGi device. They are supplied by OSGi providers and encapsulate all provider-specific details related to the location and acquisition of device driver bundles.

Method Summary

`java.lang.String[]`	`findDrivers(java.util.Dictionary props)` Return an array of driver IDs of drivers capable of attaching to a device with the given properties.
`java.io.InputStream`	`loadDriver(java.lang.String id)` Get an `InputStream` from which the driver bundle providing a driver with the giving ID can be installed.

Method Detail

findDrivers

```
public java.lang.String[] findDrivers(java.util.Dictionary props)
```

Return an array of driver IDs of drivers capable of attaching to a device with the given properties.

Parameters:

`props` - the properties of the device for which a driver is sought

Returns:

the array of driver IDs of drivers capable of attaching to a device with the given properties, or `null` if this DriverLocator does not know of any such drivers

loadDriver

```
public java.io.InputStream loadDriver(java.lang.String id)
                              throws java.io.IOException
```

Get an `InputStream` from which the driver bundle providing a driver with the giving ID can be installed.

Parameters:

`id` - the ID of the driver that needs to be installed.

Returns:

the `InputStream` from which the driver bundle can be installed

Throws:

java.io.IOException - the input stream for the bundle cannot be created

OSGi Service Gateway
Release 1.0 Final

I

implies(Permission) - Method in class org.osgi.framework.PackagePermission

Checks if the specified permission is "implied" by this object.

implies(Permission) - Method in class org.osgi.framework.AdminPermission

Checks if the specified permission is "implied" by this object.

implies(Permission) - Method in class org.osgi.framework.ServicePermission

Checks if this ServicePermission object "implies" the specified permission.

IMPORT - Static variable in class org.osgi.framework.PackagePermission

The action string "import".

installBundle(String) - Method in interface org.osgi.framework.BundleContext

Install a bundle from a location.

installBundle(String, InputStream) - Method in interface org.osgi.framework.BundleContext

Install a bundle from an InputStream.

INSTALLED - Static variable in class org.osgi.framework.BundleEvent

A bundle has been installed.

INSTALLED - Static variable in interface org.osgi.framework.Bundle

Installed state, the bundle is installed but not yet resolved.

InvalidSyntaxException - exception org.osgi.framework.InvalidSyntaxException.

Exception from the framework to indicate a filter string parameter has an invalid syntax and cannot be parsed.

InvalidSyntaxException(String, String) - Constructor for class org.osgi.framework.InvalidSyntaxException

Create the exception with the given message and the filter string which generated the exception.

L

loadDriver(String) - Method in interface org.osgi.service.device.DriverLocator

Get an `InputStream` from which the driver bundle providing a driver with the giving ID can be installed.

LOG_DEBUG - Static variable in interface org.osgi.service.log.LogService

A debugging message.

LOG_ERROR - Static variable in interface org.osgi.service.log.LogService

An error message.

LOG_INFO - Static variable in interface org.osgi.service.log.LogService

---

U

ungetService(Bundle, ServiceRegistration, Object) - Method in interface org.osgi.framework.ServiceFactory

Release a service object.

ungetService(ServiceReference) - Method in interface org.osgi.framework.BundleContext

Unget a service's service object.

uninstall() - Method in interface org.osgi.framework.Bundle

Uninstall this bundle.

UNINSTALLED - Static variable in class org.osgi.framework.BundleEvent

A bundle has been uninstalled.

UNINSTALLED - Static variable in interface org.osgi.framework.Bundle

Uninstalled state, the bundle is uninstalled and may not be used.

unregister() - Method in interface org.osgi.framework.ServiceRegistration

Unregister the service.

unregister(String) - Method in interface org.osgi.service.http.HttpService

Unregisters a previous registration done by registerServlet or registerResources.

UNREGISTERING - Static variable in class org.osgi.framework.ServiceEvent

A service is in the process of being unregistered.

update() - Method in interface org.osgi.framework.Bundle

Update this bundle.

update(InputStream) - Method in interface org.osgi.framework.Bundle

Update this bundle from an InputStream.

UPDATED - Static variable in class org.osgi.framework.BundleEvent

A bundle has been updated.

The Open Services Gateway Initiative

How This API Document Is Organized

This API (Application Programming Interface) document has pages corresponding to the items in the navigation bar, described as follows.

Overview

The Overview page is the front page of this API document and provides a list of all packages with a summary for each. This page can also contain an overall description of the set of packages.

Package

Each package has a page that contains a list of its classes and interfaces, with a summary for each. This page can contain four categories:

- Interfaces (italic)
- Classes
- Exceptions
- Errors

Class/Interface

Each class, interface, inner class and inner interface has its own separate page. Each of these pages has three sections consisting of a class/interface description, summary tables, and detailed member descriptions:

- Class inheritance diagram
- Direct Subclasses
- All Known Subinterfaces
- All Known Implementing Classes
- Class/interface declaration
- Class/interface description
- Inner Class Summary
- Field Summary
- Constructor Summary
- Method Summary
- Field Detail

- Constructor Detail
- Method Detail

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Tree (Class Hierarchy)

There is a Class Hierarchy page for all packages, plus a hierarchy for each package. Each hierarchy page contains a list of classes and a list of interfaces. The classes are organized by inheritance structure starting with `java.lang.Object`. The interfaces do not inherit from `java.lang.Object`.

- When viewing the Overview page, clicking on "Tree" displays the hierarchy for all packages.
- When viewing a particular package, class or interface page, clicking "Tree" displays the hierarchy for only that package.

Deprecated API

The Deprecated API page lists all of the API that have been deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually given. Deprecated APIs may be removed in future implementations.

Index

The Index contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

Prev/Next

These links take you to the next or previous class, interface, package, or related page.

Frames/No Frames

These links show and hide the HTML frames. All pages are available with or without frames.

Serialized Form

Each serializable or externalizable class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. While there is no link in the navigation bar, you can get to this information by going to any serialized class and clicking "Serialized Form" in the "See also" section of the class description.

This help file applies to API documentation generated using the standard doclet.

OSGi Service Gateway API Specification Release 1.0 Final

Framework	
org.osgi.framework	The OSGi Java Services Framework.

Services	
org.osgi.service.device	The OSGi Device Access Specification.
org.osgi.service.http	The OSGi HttpService Specification.
org.osgi.service.log	The OSGi LogService Specification.

All Classes

AdminPermission
Bundle
BundleActivator
BundleContext
BundleEvent
BundleException
BundleListener
Configurable
Device
Driver
DriverLocator
FrameworkEvent
FrameworkListener
HttpContext
HttpService
InvalidSyntaxException
LogEntry
LogListener
LogReaderService
LogService
NamespaceException
PackagePermission
ServiceEvent
ServiceFactory
ServiceListener
ServicePermission
ServiceReference
ServiceRegistration

org.osgi.framework

Interfaces

Bundle
BundleActivator
BundleContext
BundleListener
Configurable
FrameworkListener
ServiceFactory
ServiceListener
ServiceReference
ServiceRegistration

Classes

AdminPermission
BundleEvent
FrameworkEvent
PackagePermission
ServiceEvent
ServicePermission

Exceptions

BundleException
InvalidSyntaxException

org.osgi.service.device

Interfaces

Device
Driver
DriverLocator

org.osgi.service.http

Interfaces

HttpContext

HttpService

Exceptions

NamespaceException

org.osgi.service.log

Interfaces

LogEntry
LogListener
LogReaderService
LogService

Deprecated API

Serialized Form

Package org.osgi.framework

Class org.osgi.framework.AdminPermission implements Serializable

Class org.osgi.framework.BundleEvent implements Serializable

Class org.osgi.framework.BundleException implements Serializable

Class org.osgi.framework.FrameworkEvent implements Serializable

Class org.osgi.framework.InvalidSyntaxException implements Serializable

Class org.osgi.framework.PackagePermission implements Serializable

Class org.osgi.framework.ServiceEvent implements Serializable

Class org.osgi.framework.ServicePermission implements Serializable

Package org.osgi.service.http

Class org.osgi.service.http.NamespaceException implements Serializable

Hierarchy For All Packages

Package Hierarchies:

org.osgi.framework, org.osgi.service.device, org.osgi.service.http, org.osgi.service.log

Class Hierarchy

- class java.lang.Object
  - class java.util.EventObject (implements java.io.Serializable)
    - class org.osgi.framework.**BundleEvent**
    - class org.osgi.framework.**FrameworkEvent**
    - class org.osgi.framework.**ServiceEvent**
  - class java.security.Permission (implements java.security.Guard, java.io.Serializable)
    - class java.security.BasicPermission (implements java.io.Serializable)
      - class org.osgi.framework.**AdminPermission**
      - class org.osgi.framework.**PackagePermission**
      - class org.osgi.framework.**ServicePermission**
  - class java.lang.Throwable (implements java.io.Serializable)
    - class java.lang.Exception
      - class org.osgi.framework.**BundleException**
      - class org.osgi.framework.**InvalidSyntaxException**
      - class org.osgi.service.http.**NamespaceException**

Interface Hierarchy

- interface org.osgi.framework.**Bundle**
- interface org.osgi.framework.**BundleActivator**
- interface org.osgi.framework.**BundleContext**
- interface org.osgi.framework.**Configurable**
- interface org.osgi.service.device.**Device**
- interface org.osgi.service.device.**Driver**

- interface org.osgi.service.device.**DriverLocator**
- interface java.util.EventListener
    - interface org.osgi.framework.**BundleListener**
    - interface org.osgi.framework.**FrameworkListener**
    - interface org.osgi.service.log.**LogListener**
    - interface org.osgi.framework.**ServiceListener**
- interface org.osgi.service.http.**HttpContext**
- interface org.osgi.service.http.**HttpService**
- interface org.osgi.service.log.**LogEntry**
- interface org.osgi.service.log.**LogReaderService**
- interface org.osgi.service.log.**LogService**
- interface org.osgi.framework.**ServiceFactory**
- interface org.osgi.framework.**ServiceReference**
- interface org.osgi.framework.**ServiceRegistration**

Hierarchy For Package org.osgi.framework

Package Hierarchies:

All Packages

Class Hierarchy

- class java.lang.Object
  - class java.util.EventObject (implements java.io.Serializable)
    - class org.osgi.framework.**BundleEvent**
    - class org.osgi.framework.**FrameworkEvent**
    - class org.osgi.framework.**ServiceEvent**
  - class java.security.Permission (implements java.security.Guard, java.io.Serializable)
    - class java.security.BasicPermission (implements java.io.Serializable)
      - class org.osgi.framework.**AdminPermission**
      - class org.osgi.framework.**PackagePermission**
      - class org.osgi.framework.**ServicePermission**
  - class java.lang.Throwable (implements java.io.Serializable)
    - class java.lang.Exception
      - class org.osgi.framework.**BundleException**
      - class org.osgi.framework.**InvalidSyntaxException**

Interface Hierarchy

- interface org.osgi.framework.**Bundle**
- interface org.osgi.framework.**BundleActivator**
- interface org.osgi.framework.**BundleContext**
- interface org.osgi.framework.**Configurable**
- interface java.util.EventListener
  - interface org.osgi.framework.**BundleListener**
  - interface org.osgi.framework.**FrameworkListener**

    - interface org.osgi.framework.**ServiceListener**
- interface org.osgi.framework.**ServiceFactory**
- interface org.osgi.framework.**ServiceReference**
- interface org.osgi.framework.**ServiceRegistration**

OSGi Service Gateway
Release 1.0 Final

Hierarchy For Package org.osgi.service.http

Package Hierarchies:

All Packages

Class Hierarchy

- class java.lang.Object
  - class java.lang.Throwable (implements java.io.Serializable)
    - class java.lang.Exception
      - class org.osgi.service.http.**NamespaceException**

Interface Hierarchy

- interface org.osgi.service.http.**HttpContext**
- interface org.osgi.service.http.**HttpService**

Hierarchy For Package org.osgi.service.device

Package Hierarchies:

All Packages

Interface Hierarchy

- interface org.osgi.service.device.**Device**
- interface org.osgi.service.device.**Driver**
- interface org.osgi.service.device.**DriverLocator**

Hierarchy For Package org.osgi.service.log

Package Hierarchies:

All Packages

Interface Hierarchy

- interface java.util.EventListener
  - interface org.osgi.service.log.**LogListener**
- interface org.osgi.service.log.**LogEntry**
- interface org.osgi.service.log.**LogReaderService**
- interface org.osgi.service.log.**LogService**