Application Platform Services

User Guide

1.0.0

Tommy Svensson

Application Platform Services (APS)

Please note that this project have been going on for quite some time and have changed architecture on the way. This mostly due to limited time working on it, and partly to being a playground.

The documentation is put together from multiple sources in different maven sub modules. A lot of this documentation is currently quite out of date. There are for example code examples done before lambdas! There are also maven modules that will be removed due to no longer being relevant for the current architecture, like JPA (which I originally did just to have something that worked on any OSGi container, rather than a container specific integrated solution). JPA is not compatible with current architecture which mostly works with JSON data.

OSGi Application Platform Services - A "smorgasbord" of OSGi services that focuses on ease of use and good enough functionality for many but wont fit all. It can be seen as osgi-ee-light-and-easy. The services are of platform type: configuration, database, JPA, etc, with companion web applications for administration.

All services that require some form of administration have an admin web application for that, that plugs into the general apsadminweb admin web application.

All administrations web applications are WABs and thus require that the OSGi server supports WABs.

Another point of APS is to be as OSGi server independent as possible, but as said above the admin web applications do need support for WABs.

APS is made using basic OSGi functionality and is not using blueprint and other fancy stuff! Each bundle has an activator that does setup, creates trackers, loggers, and manually dependency injects them into the service providers it publishes.

Features

Current

- A far better service tracker that does a better job at handling services coming and going. Supports service availability wait and timeout and can be wrapped as a proxy to the service. Instead of returning null it throws an exception if no service becomes available within the timeout, and is thus much easier to handle.
- A configuration manager that extends deployed bundles by reading their configuration schema, their default configuration file, and their configuration id, and then loads and publishes an `APSConfig` instance with the bundles configuration. All active configurations are stored in a cluster (vertx/hazelcast). There will be a config web to edit configurations.
- A filesystem service that provides a persistent filesystem outside of the OSGi server. The configuration service makes use of this to store configurations. Each client can get its own filesystem area, and can´t access anything outside of its area.

Pre Setup

The Filesystem service is part of the core and used by other services. It should preferably have its filesystem root outside of the server installation. The BundleContext.getDataFile(String) returns a path within the deploy cache and is only valid for as long a a bundle is deployed. The point with the FilesystemService is to have a more permanent filesystem outside of the application server installation. To provide the FilesystemService root the following system property have to be set and available in the JVM instance:

```
aps.filesystem.root=<root>
```

How to do this differs between servers. In Glassfish you can supply system properties with its admin gui.

If this system property is not set the default root will be BundleContext.getFile(). This can work for development setup, but not for more serious installations!

After this path has been setup and the server started, all other configuration can be done in http://.../apsadminweb/.

The aps-apis bundle

This contains the general APIs for standard services that other bundles should implement. The APIs are actually one of the main points of APS. Its goal is to define trivially easy to use APIs for different things. Any needed complexity should be hidden within the API implementation and users should only have to deal with the simple API.

The project does provide a lot of implementations of the APIs. They are in 2 categories:

1. Intended to be deployed and used as is (ex: aps-vertx-provider).
2. A "default" implementation that can be copied and modified / configured to own need. Of course implementations can also be written from scratch.

For (1) there is of course nothing to stop it from being treated as (2) :-).

The aps-apis bundle also contains some base functionality like a better service tracker and a generic bundle activator that does dependency injections.

Tools

APSServiceTracker

This does the same thing as the standard service tracker included with OSGi, but does it better with more options and flexibility. One of the differences between this tracker and the OSGi one is that this throws an *APSNoServiceAvailableException* if the service is not available. Personally I think this is easier to work with than having to check for a null result. I also think that trying to keep bundles and services up are better than pulling them down as soon as one depencency goes away for a short while, for example due to redeploy of newer version. This is why APSServiceTracker takes a timeout and waits for a service to come back before failing.

Note: that in previous version APSServiceTracker did all callbacks in a separate thread. This is no longer the case, and shouldn't have been from the beginning.

There are several variants of constructors, but here is an example of one of the most used ones within the APS services:

```
APSServiceTracker<Service> tracker =
    new APSServiceTracker<Service>(context, Service.class, "20 seconds");
tracker.start();
```

Note that the third argument, which is a timeout can also be specified as an int in which case it is always in miliseconds. The string variant supports the a second word of "sec[onds]" and "min[utes]" which indicates the type of the first numeric value. "forever" means just that and requires just one word. Any other second words than those will be treated as milliseconds. The APSServiceTracker also has a set of constants for the timeout string value:

```
public static final String SHORT_TIMEOUT = "3 seconds";
public static final String MEDIUM_TIMEOUT = "30 seconds";
public static final String LARGE_TIMEOUT = "2 minutes";
public static final String VERY_LARGE_TIMEOUT = "5 minutes";
public static final String HUGE_LARGE_TIMEOUT = "10 minutes";
public static final String NO_TIMEOUT = "forever";
```

On bundle stop you should do:

```
tracker.stop(context);
```

So that the tracker unregisters itself from receiving bundle/service events.

Services and active service

The tracker tracks all instances of the service being tracked. It however have the notion of an active

service. The active service is the service instance that will be returned by allocateService() (which is internally used by all other access methods also). On startup the active service will be the first service instance received. It will keep tracking other instances comming in, but as long as the active service does not go away it will be the one used. If the active service goes away then the the one that is at the beginning of the list of the other tracked instances will become active. If that list is empty there will be no active, which will trigger a wait for a service to become available again if allocateService() is called.

Providing a logger

You can provide an APSLogger (see further down about APSLogger) to the tracker:

```
tracker.setLogger(apsLogger);
```

When available the tracker will log to this.

Tracker as a wrapped service

The tracker can be used as a wrapped service:

```
Service service = tracker.getWrappedService();
Service service = tracker.getWrappedService(boolean cacheCallsUntilServiceAvailable);
```

This gives you a proxied *service* instance that gets the real service, calls it, releases it and return the result. This handles transparently if a service has been restarted or one instance of the service has gone away and another came available. It will wait for the specified timeout for a service to become available and if that does not happen the *APSNoServiceAvailableException* will be thrown. This is of course a runtime exception which makes the service wrapping possible without loosing the possibility to handle the case where the service is not available.

The *cacheCallsUntilServiceAvailable* parameter means just that. This makes the service non blocking. Otherwise any call to a method when service is not available will result in a wait() on the thread until a service is available. When this parameter is `true` however any calls to the service before a service is available will be cached and executed later when a service is available. Do note that the tracker wrapper provides a java.lang.reflect.Proxy implementation of the service interface. Under the surface it will do an *invoke* on the actual service object and this invoke can be saved for later in a lambda. There is however a big **warning** with this: This feature will obviously only work for methods that don't provide a return value! Since method calls may possible be done in the future they cannot return any value. And no, *Future<?>* cannot be used since it blocks, and we are trying to avoid blocking here!

If you don't like this, don't use the *getWrappedService(true)*. The *.onActiveServiceAvailable(callback)* method can be used to receive the service instance when it is available.

Using the tracker in a similar way to the OSGi standard tracker

To get a service instance you do:

```
Service service = tracker.allocateService();
```

Note that if the tracker has a timeout set then this call will wait for the service to become available if it is currently not available until an instance becomes available or the timeout time is reached. It will throw *APSNoServiceAvailableException* on failure in any case.

When done with the service do:

```
tracker.releaseService();
```

Accessing a service by tracker callback

Note that the *onServiceAvailable*, *onServiceLeaving*, etc have historical reasons for the names, but

do now accept multiple calls without overwriting previous callback. The reason for this is that **APSActivator** reuses a tracker instance for tracking the same service in different classes. This allow for each class to do an *onServiceAvailable(...)* and be called back when service is available. The easiest use of **APSServiceTracker** & **APSActivator** is to inject tracker as a proxied instance of the service API, by declaring its type to be the service interface. There are however times when you need to know when a service is available and this provides that.

There are a few variants to get a service instance by callback. When the callbacks are used the actual service instance will only be allocated during the callback and then released again.

onServiceAvailable

This will result in a callback when any instance of the service becomes available. If there is more than one service instance published then there will be a callback for each.

```
tracker.onServiceAvailable(new OnServiceAvailable<Service>() {
    @Override
    public void onServiceAvailable(
        Service service,
        ServiceReference serviceReference
    ) throws Exception {
        // Do something.
    }
});
```

onServiceLeaving

This will result in a callback when any instance of the service goes away. If there is more than one service instance published the there will be a callback for each instance leaving.

```
onServiceLeaving(new OnServiceLeaving<Service>() {

    @Override
    public void onServiceLeaving(
        ServiceReference service,
        Class serviceAPI
    ) throws Exception {
        // Handle the service leaving.
    }
});
```

Note that since the service is already gone by this time you don't get the service instance, only its reference and the class representing its API. In most cases both of these parameters are irellevant.

onActiveServiceAvailable

This does the same thing as onServiceAvailable() but only for the active service. It uses the same *OnServiceAvailable* interface.

onActiveServiceLeaving

This does the same thing as onServiceLeaving() but for the active service. It uses the same *OnServiceLeaving* interface.

withService

Runs the specified callback providing it with a service to use. This will wait for a service to become available if a timeout has been provided for the tracker.

Don't use this in an activator start() method! onActiveServiceAvailable() and onActiveServiceLeaving() are safe in a start() method, this is not!

```
tracker.withService(new WithService<Service>() {
    @Override
    public void withService(
```

```
                Service service,
                Object... args
            ) throws Exception {
                // do something here.
            }
        }, arg1, arg2);
```

If you don't have any arguments this will also work:

```
        tracker.withService(new WithService<Service>() {
            @Override
            public void withService(
                Service service
            ) throws Exception {
                // do something here
            }
        });
```

withServiceIfAvailable

This does the same as withService(...) but without waiting for a service to become available. If the service is not available at the time of the call the callback will not be called. No exception is thrown by this!

withAllAvailableServices

This is used exactly the same way as withService(...), but the callback will be done for each tracked service instance, not only the active.

onTimeout (since 0.9.3)

This allows for a callback when the tracker times out waiting for a service. This callback will be called just before the *APSNoServiceAvailableException* is about to be thrown.

```
    tracker.onTimeout(new OnTimeout() {
        @Override
        public void onTimeout() {
            // do something here
        }
    });
```

APSLogger

This provides logging functionality. The no args constructor will log to System.out by default. The OutputStream constructor will logg to the specified output stream by default.

The APSLogger can be used by just creating an instance and then start using the info(...), error(...), etc methods. But in that case it will only log to System.out or the provided OutputStream. If you however do this:

```
    APSLogger logger = new APSLogger();
    logger.start(context);
```

then the logger will try to get hold of the standard OSGi LogService and if that is available log to that. If the log service is not available it will fallback to the OutputStream.

If you call the *setServiceRefrence(serviceRef);* method on the logger then information about that service will be provied with each log.

APSActivator

This is a BundleActivator implementation that uses annotations to register services and inject tracked services. Any bundle can use this activator by just importing the *se.natusoft.osgi.aps.activator* and

se.natusoft.osgi.aps.activator.annotation packages.

This is actually a rather trivial class that just scans the bundle for classes and inspects all classes for annotations and act on them.

Please note that it does *class.getDeclaredFields()* and *class.getDeclaredMethods()*! This means that it will only see the bottom class of an inheritance hiearchy!

The following annotations are available:

@OSGiServiceProvider - This should be specified on a class that implements a service interface and should be registered as an OSGi service. *Please note* that the first declared implemented interface is used as service interface unless you specify serviceAPIs={Svc.class, ...}.

```
public @interface OSGiProperty {
    String name();
    String value();
}

public @interface OSGiServiceInstance {

    /** Extra properties to register the service with. */
    OSGiProperty[] properties() default {};

    /**
     * The service API to register instance with. If not specified the first
     * implemented interface will be used.
     */
    Class[] serviceAPIs() default {};
}

public @interface OSGiServiceProvider {
    /** Extra properties to register the service with. */
    OSGiProperty[] properties() default {};

    /**
     * The service API to register instance with. If not specified the first
     * implemented interface will be used.
     */
    Class[] serviceAPIs() default {};

    /**
     * This can be used as an alternative to properties() and also supports
     * several instances.
     */
    OSGiServiceInstance[] instances() default {};

    /**
     * An alternative to providing static information. This class will be
     * instantiated if specified and provideServiceInstancesSetup() will
     * be called to provide implemented service APIs, service properties,
     * and a service instance. In this last, it differs from
     * instanceFactoryClass() since that does not provide an instance.
     * This allows for more easy configuration of each instance.
     */
    Class<? extends APSActivatorServiceSetupProvider>
        serviceSetupProvider()
        default APSActivatorServiceSetupProvider.class;

    /**
     * This can be used as an alternative and will instantiate the
     * specified factory class which will deliver one set of
     * Properties per instance.
     */
    Class<? extends APSActivator.InstanceFactory> instanceFactoryClass()
        default APSActivator.InstanceFactory.class;

    /**
     * If true this service will be started in a separate thread.
     * This means the bundle start will continue in parallel and
     * that any failures in startup will be logged, but will
     * not stop the bundle from being started. If this is true
     * it wins over required service dependencies of the service
     * class. Specifying this as true allows you to do things that
```

```
         * cannot be done in a bunde activator start method, like
         * calling a service tracked by APSServiceTracker, without
         * causing a deadlock.
         */
        boolean threadStart() default false;
    }
```

Do note that for the *serviceSetupProvider()* another solution is to use the *@BundleStart* (see below) and just create instances of your service and register them with the BundleContext. But if you use *@OSGiServiceProvider* to instantiate and register other "one instance" services, then using *serviceSetupProvider()* would look a bit more consistent.

@OSGiService - This should be specified on a field having a type of a service interface to have a service of that type injected, and continuously tracked. Any call to the service will throw an APSNoServiceAvailableException (runtime) if no service has become available before the specified timeout. It is also possible to have APSServiceTracker as field type in which case the underlying configured tracker will be injected instead.

If *required=true* is specified and this field is in a class annotated with *@OSGiServiceProvider* then the class will not be registered as a service until the service dependency is actually available, and will also be unregistered if the tracker for the service does a timeout waiting for a service to become available. It will then be reregistered again when the dependent service becomes available again. Please note that unlike iPOJO the bundle is never stopped on dependent service unavailability, only the actual service is unregistered as an OSGi service. A bundle might have more than one service registered and when a dependency that is only required by one service goes away the other service is still available.

The non blocking variant of *APSServiceTracker.getWrappedService(true)* as described above can also be achieved with this annotation by setting *nonBlocking = true*.

```
    public @interface OSGiService {

        /**
         * The timeout for a service to become available. Defaults
         * to 30 seconds.
         */
        String timeout() default "30 seconds";

        /**
         * Any additional search criteria. Should start with
         * '(' and end with ')'. Defaults to none.
         */
        String additionalSearchCriteria() default "";

        /**
         * This should specify a Class implementing
         * APSActivatorSearchCriteriaProvider. If specified it will
         * be used instead of additionalSearchCriteria() by
         * instantiating the Class and calling its method to get
         * a search criteria back. This allows for search criteria
         * coming from configuration, which a static annotation String
         * does not.
         */
        Class<? extends APSActivatorSearchCriteriaProvider>
            searchCriteriaProvider()
            default APSActivatorSearchCriteriaProvider.class;

        /**
         * If set to true the service using this service will not
         * be registered until the service becomes available.
         */
        boolean required() default false;

        /**
         * If this is set to true and a proxied implementation of the service is injected rather than
        the tracker directly
         * then any call made to the proxy will be cached if the service is not available and then
        later run when the
         * service becomes available. This of course means that methods returning a value will always
        return null when
```

```
         * service is not currently available since the real call will be made in the future.
        Returning a Future instead
         * in this case does not work since 'Future's are blocking, and we try to avoid blocking here.
         *
         * __YOU HAVE TO BE VERY CAREFUL WHEN SETTING THIS TO TRUE! NO CALLS RETURNING A VALUE!__
         *
         * The point of this is to be non blocking. By default with a proxied implementation
        tracker.allocateService() will
         * be called, and this blocks waiting for the service to become available if it is not.
         *
         * @return true or false (default).
         */
        boolean nonBlocking() default false;
    }
```

@Managed - This will have an instance managed and injected. There will be a unique instance for each name specified with the default name of "default" being used if none is specified. There are 2 field types handled specially: BundleContext and APSLogger. A BundleContext field will get the bundles context injected. For an APSLogger instance the 'loggingFor' annotation property can be specified. Please note that any other type must have a default constructor to be instantiated and injected!

```
    public @interface Managed {

        /**
         * The name of the instance to inject. If the same is used
         * in multiple classes the same instance will be injected.
         */
        String name() default "default";

        /**
         * A label indicating who is logging. If not specified the
         * bundle name will be used. This is only
         * relevant if the injected type is APSLogger.
         */
        String loggingFor() default "";
    }
```

@Schedule - Schedules a Runnable using a ScheduledExecutionService. @Schedule is handled after all injections have been done.

```
    @Retention(RetentionPolicy.RUNTIME)
    @Target(ElementType.FIELD)
    public @interface Schedule {

        /**
         * The defined executor service to schedule this on. This should be the name of it. If left
        blank an internal
         * ScheduledExecutorService will be used.
         */
        String on() default "";

        /** The amount of time to wait for the (first) execution. */
        long delay();

        /** If specified how long to wait between runs. */
        long repeat();

        /** The time unit used for the above values. Defaults to seconds. */
        TimeUnit timeUnit() default TimeUnit.SECONDS;

        /** Possibility to affect the size of the thread pool when such is created internally for
        this (on="..." not provided!). */
        int poolSize() default 2;
    }
```

@BundleStart - This should be used on a method and will be called on bundle start. The method should take no arguments. If you need a BundleContext just inject it with *@Managed*. The use of this annotation is only needed for things not supported by this activator. Please note that a method annotated with this annotation can be static (in which case the class it belongs to will not be

instantiaded). You can provide this annotation on as many methods in as many classes as you want. They will all be called (in the order classes are discovered in the bundle).

```
public @interface BundleStart {

    /**
     * If true the start method will run in a new thread.
     * Any failures in this case will not fail
     * the bundle startup, but will be logged.
     */
    boolean thread() default false;
}
```

@BundleStop - This should be used on a method and will be called on bundle stop. The method should take no arguments. This should probably be used if *@BundleStart* is used. Please note that a method annotated with this annotation can be static!

```
public @interface BundleStop {}
```

TODO: Since APS is nowmore built on top of vertx, servlets no longer need to be supported. Vaadin (traditional) is not supported either due to requiring servlets. Vaadin web components that works with Angular and React is however supported.

Usage as BundleActivator

The *APSActivator* class has 2 constructors. The default constructor without arguments are used for BundleActivator usage. In this case you just specify this class as your bundles activator, and then use the annotations described above. Thats it!

Other Usage

Since the activator usage will manage and create instances of all annotated classes this will not always work in all situations. One example is web applications where the web container is responsible for creating servlets. If you specifiy APSActivator as an activator for a WAB bundle and then use the annotations in a servlet then APSActivator will have a managed instance of the servlet, but it will not be the same instance as the web contatiner will run.

Therefore APSActivator has another constructor that takes a vararg of instances: `public APSActivator(Object... instances)`. There is also a `public void addManagedInstance(Object instance)` method. These allow you to add an already existing instance to be managed by APSActivator. In addition to the provided existing instances it will still scan the bundle for classes to manage. It will however not double manage any class for which an existing instance of has already been provided. Any annotated class for which existing instances has not been provided will be instantiated by APSActivator.

Please note that if you create an instance of APSActivator in a servlet and provide the servlet instance to it and start it (you still need to do *start(BundleContext)* and *stop(BundleContext)* when used this way!), then you need to catch the close of the servlet and do *stop* then.

There are 2 support classes:

- [APSVaadinWebTools]: APSVaadinOSGiApplication - This is subclassed by your Vaading application.
- [APSWebTools]: APSOSGiSupport - You create an instance of this in a servlet and let your servlet implement the *APSOSGiSupportCallbacks* interface which is then passed to the constructor of APSOSGiSupport.

Both of these creates and manages an APSActivator internally and catches shutdown to take it down. They also provide other utilities like providing the BundleContext. See *APSWebTools* for more information.

Be warned that this is currently very untested! No APS code uses this yet.

APSContextWrapper

This provides a static wrap(...) method:

```
Service providedService = APSContextWrapper.wrap(serviceProvider, Service.class);
```

where *serviceProvider* is an instance of a class that implements *Service*. The resulting instance is a java.lang.reflect.Proxy implementation of *Service* that ensures that the *serviceProvider* ClassLoader is the context class loader during each call to all service methods that are annotated with @APSRunInBundlesContext annotation in *Service*. The wrapped instance can then be registered as the OSGi service provider.

Normally the threads context class loader is the original service callers context class loader. For a web application it would be the web containers context class loader. If a service needs its own bundles class loader during its execution then this wrapper can be used.

ID generators

There is one interface:

```
/**
 * This is a generic interface for representing IDs.
 */
public interface ID extends Comparable<ID> {

    /**
     * Creates a new unique ID.
     *
     * @return A newly created ID.
     */
    public ID newID();

    /**
     * Tests for equality.
     *
     * @param obj The object to compare with.
     *
     * @return true if equal, false otherwise.
     */
    @Override
    public boolean equals(Object obj);

    /**
     * @return The hash code.
     */
    @Override
    public int hashCode();

}
```

that have 2 implementations:

• IntID - Produces int ids.
• UUID - Produces java.util.UUID Ids.

Javadoc

The javadoc for this can be found at http://apidoc.natusoft.se/APSToolsLib/.

APSFilesystemService

This provides a filesystem for writing and reading files. This filesystem resides outside of the OSGi server and is for longterm storage, which differs from BundleContext.getDataFile() which resides within bundle deployment. The APSFilesystemService also does not return a File object! It priovides a file area for each unique owner name that is accessed through an API that cannot navigate nor access any files outside of this area. The ”owner” name should be either an application name or a bundle name if it is only used by one bundle.

The APSConfigService uses the APSFilesystemService to store its configurations.

Setup

The *aps.filesystem.root* system property must be set to point to a root where this service provides its file areas. This is either passed to the JVM at server startup or configured withing the server. Glassfish allows you to configure properties within its admin gui. Virgo does not. If this is not provided the service will use BundleContext.getDataFile(".") as the root, which will work for testing and playing around, but should not be used for more serious purposes since this is not a path with a long term availability.

The service

The service allows you to create or get an APSFilesystem object. From that object you can create/read/delete directories (represented by APSDirectory) and files (represented by APSFile). You can get readers, writers, input streams and output streams from files. All paths are relative to the file area represented by the APSFilesystem object.

The javadoc for the APSFilesystemService.

The APIs for this service

public *interface* **APSFile** [se.natusoft.osgi.aps.api.core.filesystem.model] {

This represents a file in an *APSFilesystemService* provided filesystem. It provides most of the API of *java.io.File* but is not a *java.io.File*! It never discloses the full path in the host filesystem, only paths relative to its *APSFilesystem* root.

Use the createInputStream/OutputStream/Reader/Writer to read and write the file.

InputStream createInputStream() throws IOException

Creates a new *InputStream* to this file.

Throws

IOException- on failure

OutputStream createOutputStream() throws IOException

Creates a new *OutputStream* to this file.

Throws

IOException- on failure

Reader createReader() throws IOException

Creates a new *Reader* to this file.

Throws

IOException- on failure

Writer createWriter() throws IOException

Creates a new *Writer* to this file.

Throws

IOException- on failure

Properties loadProperties() throws IOException

If this file denotes a properties file it is loaded and returned.

Throws

IOException- on failure or if it is not a properties file.

void saveProperties(Properties properties) throws IOException

If this file denotes a properties file it is written with the specified properties.

Parameters

properties- The properties to save.

Throws

IOException- on failure or if it is not a properties file.

APSDirectory toDirectory()

If this *APSFile* represents a directory an *APSDirectory* instance will be returned. Otherwise *null* will be returned.

APSFile getAbsoluteFile()

See

java.io.File.getAbsoluteFile()

String getAbsolutePath()

Returns the absolute path relative to filesystem root.

APSFile getCanonicalFile() throws IOException

See

java.io.File.getCanonicalFile()

String getCanonicalPath() throws IOException

See

java.io.File.getCanonicalPath()

String getParent()

See

java.io.File.getParent()

APSDirectory getParentFile()

See

java.io.File.getParentFile()

String getPath()

See

java.io.File.getPath()

boolean renameTo(APSFile dest)

See

java.io.File.renameTo(File)

String getName()

See

java.io.File.getName()

boolean canRead()

See

java.io.File.canRead()

boolean canWrite()

See

java.io.File.canWrite()

boolean exists()

See

java.io.File.exists()

boolean exists(String name)

Checks if the named file/directory exists.

Returns

true or false.

Parameters

name- The name to check.

boolean existsAndNotEmpty(String name)

Checks if the named file exists and is not empty.

Returns

true or false.

Parameters

name- The name of the file to check.

boolean isDirectory()

See

java.io.File.isDirectory()

boolean isFile()

See

java.io.File.isFile()

boolean isHidden()

See

java.io.File.isHidden()

long lastModified()

See

java.io.File.lastModified()

long length()

See

java.io.File.length()

boolean createNewFile() throws IOException

See

java.io.File.createNewFile()

boolean delete()

See

java.io.File.delete()

void deleteOnExit()

See

java.io.File.deleteOnExit()

String toString()

Returns a string representation of this *APSFile*.

File toFile()

This API tries to hide the real path and don't allow access outside of its root, but sometimes you just need the real path to pass on to other code requiring it. This provides that. Use it only when needed!

Returns

A File object representing the real/full path to this file.

}

---

public *interface* **APSFilesystem** [se.natusoft.osgi.aps.api.core.filesystem.model] {

This represents an *APSFilesystemService* filesystem.

APSDirectory getDirectory(String path) throws IOException

Returns a folder at the specified path.

Parameters

path- The path of the folder to get.

Throws

IOException- on any failure, specifically if the specified path is not a folder or doesn't exist.

APSFile getFile(String path)

Returns the file or folder of the specified path.

Parameters

path- The path of the file.

APSDirectory getRootDirectory()

Returns the root directory.

}

---

public *interface* **APSDirectory** extends APSFile [se.natusoft.osgi.aps.api.core.filesystem.model] {

This represents a directory in an *APSFilesystem*.

Use this to create or get directories and files and list contents of directories.

Personal comment: I do prefer the term "folder" over "directory" since I think that is less ambiguous, but since Java uses the term "directory" I decided to stick with that name.

APSDirectory createDir(String name) throws IOException

Returns a newly created directory with the specified name.

Parameters

name- The name of the directory to create.

Throws

IOException- on any failure.

APSDirectory createDir(String name, String duplicateMessage) throws IOException

Returns a newly created directory with the specified name.

Parameters

name- The name of the directory to create.

duplicateMessage- The exception messaging if directory already exists.

Throws

IOException- on any failure.

APSFile createFile(String name) throws IOException

Creates a new file in the directory represented by the current *APSDirectory*.

Parameters

name- The name of the file to create.

Throws

IOException- on failure.

APSDirectory getDir(String dirname) throws FileNotFoundException

Returns the specified directory.

Parameters

dirname- The name of the directory to enter.

Throws

FileNotFoundException- on failure

APSFile getFile(String name)

Returns the named file in this directory.

Parameters

name- The name of the file to get.

void recursiveDelete() throws IOException

Performs a recursive delete of the directory represented by this *APSDirectory* and all subdirectories and files.

Throws

IOException- on any failure.

String[] list()

See

java.io.File.list()

APSFile[] listFiles()

See

java.io.File.listFiles()

}

---

public *interface* **APSFilesystemService** [se.natusoft.osgi.aps.api.core.filesystem.service] {

This provides a filesystem for use by services/applications. Each filesystem has its own root that cannot be navigated outside of.

Services or application using this should do something like this in their activators:

```
APSFilesystemService fss;
APSFilesystem fs;

fss.getFilesystem("my.file.system", (result) -> {
    if (result.success()) {
        fs = result.result();
    }
});
```

void getFilesystem(String owner, APSHandler<APSResult<APSFilesystem>> handler)

Returns the filesystem for the specified owner. If the filesystem does not exist it is created.

Parameters

owner- The owner of the filesystem or rather a unique identifier of it.

handler- Called with the filesystem.

Throws

APSIOException- on failure.

void deleteFilesystem(String owner, APSHandler<APSResult<Void>> handler)

Removes the filesystem and all files in it.

Parameters

owner- The owner of the filesystem to delete.

Throws

APSIOException- on any failure.

}

---

aps-core-lib

Configuration for Bundles

APSConfigLoader

This is a trivially easy way of getting configuration. Just do:

```
Map<String, Object> config = APSConfigLoader.get("config-id")
```

Where there are 2 resource files under `apsconfig`:

```
apsconfig/
    (config-id)-schema.json
    default-(config-id)-config.json
```

To provide a configuration that differs from bundle default, package:

```
apsconfig/
    (config_id)-config.json
```

in a jar file and include in APS-Runtime under *dependencies*. This will override the default config file delivered with bundle. It is possible to include multiple bundles config files in the same jar of course.

Do note that if `(config-id)-schema.json` is provided then the configuration file used will be validated against it. If no schema file is provided by the bundle then no validation will be done and whatever is in the config file will be loaded without error. It must of course be a JSON file or it will fail!

MapJsonDocValidator

This takes a schema (made up of a `Map<String, Object>` , see below) and another `Map<String, Object>` representing the JSON. So the catch here is that you need a JSON parser that allows you to get the content as a Map. The Vertx JSON parser does. This uses `Map` since it is generic, does not need to hardcode dependency on a specific parser, and maps are very easy to work with in Groovy.

Useage

```
 private Map<String, Object> schema = [
        "header_?": "Contains meta data about the message.",
        "header_1": [
                "type_?"      : "The type of the message. Currently only 'service'.",
                "type_1"      : "service",
                "address_?"   : "The address of the sender.",
                "address_1"   : "?aps\\.admin\\..*",
                "classifier_1": "?public|private"
        ],
        "body_1"  : [
                "action_1": "get-webs"
        ],
        "reply_0": [
                "webs_1": [
                        [
                                "name_1": "?.*",
                                "url_1": "?^https?://.*",
                                "no1_0": "#1-100",
                                "no2_0": "#<=10",
                                "no3_0": "#>100",
                                "no4_0": "#1.2-3.4"
                        ]
                ]
        ]
] as Map<String, Object>

private MapJsonDocValidator verifier = new MapJsonDocValidator( validStructure: schema )

...
```

```
        verifier.validate(myJsonMap)
```

This will throw a runtime exception on validation failure, specifically APSValidationException.

Note that there is also a special feature for defining a simple dynamic key=>value map where the key is defined with a regexp and the value can be a regexp, string, number, or boolean. This is done be defining a map with only one entry. Example:

```
    private Map<String, Object> schema = [
        "nameAddress": [
            "?([a-z]|[0-9]|_|-)+": "?[0-9,.]+"
        ]
    ]
```

In this case `nameAdress` can contain any number of entries as long as each entry have a key containging a-z or 0-9 or _ or - and there must be at least one character, and the value only contains numbers and dots.

Note that when the key is a regexp (starts with '?') then there can be no more rule for this submap!

Schema

Keys

<key>_0 - The key is optional.

<key>_1 - The key is required.

<key>_? - A description of the key. MapJsonSchemaMeta extracts this information. It is intended for editors editing data of a file validated by a schema. This should provide help information about the value. Since APS uses the MapJsonDocSchemaValidator for configurations and is intended to have web guis for editing configuration this is intended to provide information about configuration fields.

?regexp - special handling. See usage above.

Values

"?regexp"

The '?' indicates that the rest of the value is a regular expression. This regular expression will be applied to each value.

"\<hash\>\<range\>"

This indicates that this is a number and defines the number range allowed. The following variants are available:

"#from-to" : This specifies a range of allowed values, from lowest to highest.

"#<=num" : This specifies that the numeric value must be less than or equal to the specified number.

"#>=num" : This specifies that the numeric value must be larger than or equal to the specified number.

"#<num" : This specifies that the numeric value must be less than the specified number.

"#>num" : This specifies that the numeric value must be larger than the specified number.

Note: Both floating point numbers and integers are allowed.

"bla"

This requires values to be exactly "bla".

Example

```
Map<String, Object> myJsonObject = JSON.readJsonAsMap( myJsonStream,
    jsonErrorHandler)

...

Map<String, object> schema = JSON.readJsonAsMap(schemaStream, jsonErrorHandler)
MapJsonDocValidator jsonValidator = new MapJsonDocValidator( validstructure: schema )

jsonValidator.validate( myJsonObject )
```

MapJsonSchemaMeta

This class scans a MapJson schema as defined by MapJsonDocValidator and extracts a list of MapJsonEntryMeta instances for each value in a MapJson structure living up to the schema.

From these the following can be resolved:

- The name of a value.
- The type of a value.
- Is the value required ?
- The constraints of the value. If this starts with '?' then the rest is a regular expression. If not the value is a constant, that is, the value has to be exactly as the constraint string.
- A description of the value.

This is not used by the MapJsonDocValidator when validating! This is intended for GUI configuration editors to use to build a configuration GUI producing valid configurations.

Usage:

```
Map<String, Object> schema
...
new MapJsonSchemaMeta(schema).mapJsonEntryMetas.each { MapJsonEntryMeta mjem -> ... }
```

StructMap

This wraps a structured Map that looks like a JSON document, containing Map, List, and other 'Object's as values.

A key is a String containing branches separated by '.' characters for each sub structure.

It provides a method to collect all value referencing keys in the map structure with full key paths.

It provides a lookup method that takes a full value key path and returns a value.

Note that since this delegates to the wrapped Map the class is also a Map when compiled!

Here is an example (in Groovy) that shows how to lookup and how to use the keys:

```
StructMap smap = new StructMap<>(
        [
                header: [
                        type      : "service",
                        address   : "aps.admin.web",
                        classifier: "public",
                        enabled   : true
                ],
                body  : [
                        action: "get-webs"
                ],
                reply : [
                        webs: [
                                [
                                        name: "ConfigAdmin",
```

```
                                        url : "http://localhost:8080/aps/ConfigAdminWeb",
                                    ],
                                    [
                                        name: "RemoteServicesAdmin",
                                        url : "https://localhost:8080/aps/RemoteSvcAdmin"
                                    ]
                                ]
                            ]
                    ] as Map<String, Object>
    ) as StructMap

    assert smap.lookup( "header.type" ).toString() == "service"
    assert smap.lookup( "header.address" ).toString() == "aps.admin.web"
    assert smap.lookup( "header.classifier" ).toString() == "public"
    assert smap.lookup( "body.action" ).toString() == "get-webs"
    assert smap.lookup( "reply.webs.[0].name") == "ConfigAdmin"
    assert smap.lookup( "reply.webs.[1].name") == "RemoteServicesAdmin"
    assert smap.lookup( "reply.webs.[1].url") == "https://localhost:8080/aps/RemoteSvcAdmin"

    smap.withAllKeys { String key ->
        println "${key}"
    }

    // will produce:
    header.type
    header.address
    header.classifier
    header.enabled
    body.action
    reply.webs.[2].name
    reply.webs.[2].url
```

Note that the values are API-wise of type Object! This is because it can be anything, like a String, Map, List, Number (if you stick to JSON formats) or any other type of value you put in there.

Also note the indexes in the keys in the example. It is not "webs[0]" but "webs.[0]"! The index is a reference name in itself. The keys returned by getAllKeys() have a number between the '[' and the ']' for List entries. This number is the number of entries in the list. The StructPath class (used by this class) can be used to provide array size of an array value.

APSBus

Since there are many types of busses out there and that APS is based on Vert.x with its own bus (*EventBus*) APS provides a very simple, generic bus API called **APSBus**. It should be documented elsewhere in this document.

This bundle contains an implementation of APSBus. The *APSBus* implementation just tracks all published `APSBusRouter` implementations. Each `APSBusRouter` implementations must also provide a resource file in *aps/bus/routers* with the name of each bus router, one per line. APSBus will find all these and wait for them to become available as services before publishing itself as a service.

If an `APSBusRouter` implementation does not do this, then it is possible that APSBus will not see it. It is also possible that it will se it, but miss another implementation instead. This due to it actually not knowing the available implementations nor their names. It just count the entries in all found *routers* files, and waits for that amount of `APSBusRouter` services to be published. The names are just for show and are logged on startup.

Licenses

Project License

Apache Software License version 2.0

Third Party Licenses

OSGi Specification License version 2.0

The following third party products are using this license:

• org.osgi.compendium-4.2.0-null
• org.osgi.core-4.2.0-null

Apache Software License version 2.0

The following third party products are using this license:

• jackson-jr-all-2.9.6
• groovy-3.0.2
• vertx-core-3.8.0
• vertx-lang-groovy-3.8.0
• vertx-auth-common-3.8.0
• vertx-web-3.8.0
• vertx-amqp-bridge-3.8.0
• vertx-amqp-client-3.8.0
• vertx-hazelcast-3.8.0
• hazelcast-3.6.3
• jackson-core-2.12.3
• jackson-annotations-2.12.3
• jackson-databind-2.12.3
• netty-codec-4.1.19.Final
• netty-codec-http-4.1.19.Final
• netty-codec-http2-4.1.19.Final
• netty-codec-dns-4.1.19.Final
• netty-codec-socks-4.1.19.Final
• netty-handler-4.1.19.Final
• netty-handler-proxy-4.1.19.Final
• netty-resolver-4.1.19.Final
• netty-resolver-dns-4.1.19.Final
• netty-buffer-4.1.19.Final
• netty-common-4.1.19.Final
• netty-transport-4.1.19.Final
• geronimo-jta_1.1_spec-1.1.1
• commons-net-3.5

Eclipse Public License - v version 1.0

Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and b) in the case of each subsequent Contributor: i) changes to the Program, and ii)

additions to the Program; where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

1. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

1. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone

and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

1. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

1. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

1. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

OSGi Specification License, Version 2.0.

License Grant

OSGi Alliance (“OSGi”) hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under OSGi’s applicable intellectual property rights to view, download, and reproduce this OSGi Specification (“Specification”) which follows this License Agreement (“Agreement”). You are not authorized to create any derivative work of the Specification. However, to the extent that an implementation of the Specification would necessarily be a derivative work of the Specification, OSGi also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Specification that: (i) fully implements the Specification including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Specification. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Specification, does not receive the benefits of this license, and must not be described as an implementation of the Specification. An implementation of the Specification must not

claim to be a compliant implementation of the Specification unless it passes the OSGi Compliance Tests for the Specification in accordance with OSGi processes. “OSGi Name Space” shall mean the public class or interface declarations whose names begin with “org.osgi" or any recognized successors or replacements thereof.

OSGi Participants (as such term is defined in the OSGi Intellectual Property Rights Policy) have made non-assert and licensing commitments regarding patent claims necessary to implement the Specification, if any, under the OSGi Intellectual Property Rights Policy which is available for examination on the OSGi public web site (www.osgi.org).

No Warranties and Limitation of Liability

THE SPECIFICATION IS PROVIDED "AS IS," AND OSGi AND ANY OTHER AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. OSGi AND ANY OTHER AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SPECIFICATION OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

Covenant Not to Assert

As a material condition to this license you hereby agree, to the extent that you have any patent claims which are necessarily infringed by an implementation of the Specification, not to assert any such patent claims against the creation, distribution or use of an implementation of the Specification.

General

The name and trademarks of OSGi or any other Authors may NOT be used in any manner, including advertising or publicity pertaining to the Specification or its contents without specific, written prior permission. Title to copyright in the Specification will at all times remain with OSGi.

No other rights are granted by implication, estoppel or otherwise.

Apache License version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to

software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

1. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
2. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
3. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
   3.1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
   3.2. You must cause any modified files to carry prominent notices stating that You changed the files; and
   3.3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
   3.4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever

such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

4. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
5. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
6. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
7. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
8. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

```
    Copyright [yyyy] [name of copyright owner]

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.
```