

Application Platform Services

User Guide

1.0.0

Tommy Svensson

Copyright © 2013 Natusoft AB

Application Platform Services (APS)	1
<i>Features</i>	<i>1</i>
Current	1
Planned	1
What is new in	1
<i>Pre Setup</i>	<i>1</i>
The aps-apis bundle	3
<i>Exceptions</i>	<i>3</i>
<i>Tools</i>	<i>3</i>
APSServiceTracker	3
Services and active service	4
Providing a logger	4
Tracker as a wrapped service	4
Using the tracker in a similar way to the OSGi standard tracker	5
Accessing a service by tracker callback	5
onServiceAvailable	5
onServiceLeaving	5
onActiveServiceAvailable	6
onActiveServiceLeaving	6
withService	6
withServiceIfAvailable	6
withAllAvailableServices	6
onTimeout (since 0.9.3)	6
APSLogger	7
APSAActivator	7
Usage as BundleActivator	11
Other Usage	11
APSAActivatorPlugin	11
APSContextWrapper	12
ID generators	12
Javadoc	13
<i>APSCfgManager</i>	<i>13</i>
MapJSON	14
APSFilystemService	15
<i>Setup</i>	<i>15</i>
<i>The service</i>	<i>15</i>
<i>The APIs for this service</i>	<i>15</i>
APSJSONLib	22
<i>Changes</i>	<i>22</i>
0.10.0	22
<i>APIs</i>	<i>22</i>
aps-core-lib	40
<i>MapJsonDocValidator</i>	<i>40</i>
Useage	40
Schema	40
Keys	40
Values	41
"?regexp"	41
"<hash><range>"	41
"bla"	41
Example	41
<i>MapJsonSchemaMeta</i>	<i>41</i>
<i>StructMap</i>	<i>42</i>
APSJPAService	44
<i>APIs</i>	<i>44</i>
APS RabbitMQ Message Service Provider	46
<i>APSMessagingService API</i>	<i>46</i>
<i>aps-vertx-event-bus-messaging-provider</i>	<i>46</i>
Lookup	46
Web Manager	47
<i>Components</i>	<i>47</i>
<i>APSEventBus</i>	<i>47</i>
APSLocalEventBusRouter	47
APSVtxEventBusRouter	47

<i>Messaging</i>	47
Addresses	47
Address Strategy	47
Routing Strategy	48
Messages	48
General	48
--> Avail	49
<-- Gui	49
--> Create	49
Licenses	50
<i>Project License</i>	50
<i>Third Party Licenses</i>	50
<i>Eclipse Public License - v version 1.0</i>	50
<i>OSGi Specification License, Version 2.0.</i>	53
<i>Eclipse Public License -v 1.0 version 1.0</i>	54
<i>Apache License version 2.0, January 2004</i>	55
APPENDIX: How to apply the Apache License to your work.	57

Application Platform Services (APS)

OSGi Application Platform Services - A "smorgasbord" of OSGi services that focuses on ease of use and good enough functionality for many but won't 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 `APSSConfig` 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.
- A platform service that simply identifies the local installation and provides a description of it. It is basically a read only service that provides configured information about the installation.
- A JPA service that is easier and more clearly defined than the osgi-ee JPA API, and allows for multiple JPA contexts. It works as an extender picking up persistence.xml whose defined persistence unit name can then be looked up using the service. A client can only lookup its own persistence units. It is based on OpenJPA.
- An administration web service to which administration web applications can register themselves with an url and thus be available in the ../apsadminweb admin gui.

Planned

- An implementation of the standard OSGi LogService since not all servers provide one.
- A log viewer web application supporting regular expression filters on log information and a live log view.
- Anything else relevant I come up with and consider fun to do :-).

What is new in

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 as 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 has 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. These were previously (<1.0.0) in a separate bundle called aps-tools-lib, but the relationships between that bundle and this finally became too circular so I merged aps-tools-lib into aps-apis.

Exceptions

APS defines the following base exceptions:

- *APSException* - The base exception extended by all other APS exceptions. This is a RuntimeException! APS uses no checked exceptions.
- *APSConfigException* - Thrown on configuration problems.
- *APSIOWException* - General IO problems.
- *APSIOWTimeoutException* - IO timeouts.
- *APSPersistenceException* - Persistence problems.
- *APSResourceNotFoundException* - Resource problems.
- *APSStartException* - Bundle startup problems.
- *APSUnsupportedException* - Thrown when something is not supported by a bundle or API implementation. Should be clearly documented.
- *APSValidationException* - Thrown on validation failures.

The reason for these are to be able to separate APS specific exceptions from other exceptions if needed.

Also note that much of the APS APIs are reactive with a result callback to an `APSHandler`. These do not throw exceptions, but provides an exception instance in a failure result.

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 dependency 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 milliseconds. 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 coming 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 *APSServiceUnavailableException* 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 possibly 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 *APSNServiceAvailableException* 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 irrelevant.

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 log 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 *setServiceReference(serviceRef)* method on the logger then information about that service will be provided 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 hierarchy!

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 bundle 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.

@APSExternalizable, @APSRemoteService - These 2 annotations are synonyms and have no properties. They should be used on a service implementation class. When either of these are specified the "aps-externalizable=true" property will be set when the service is registered with the OSGi container. The APSExternalProtocolExtender will react on this property and make the service externally accessible.

@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 *APSServiceUnavailableException* (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 "";
    }

```

@ExecutorSvc - This should always be used in conjunction with **@Managed**! This also assumes that the annotated field is of type **ExecutorService** or **ScheduledExecutorService**. This annotation provides some configuration for the **ExecutorService** that will be injected.

```

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

    enum ExecutorType {
        FixedSize,
        WorkStealing,
        Single,
        Cached,
        Scheduled,
        SingleScheduled
    }

    /** This is loosely the number of concurrent threads. */
    int parallelism() default 10;

    /** The type of ExecutorService wanted. */
    ExecutorType type() default ExecutorType.FixedSize;

    /** If true the created ExecutorService will be wrapped with a delegate that disallows
    configuration. */
    boolean unConfigurable() default false;
}

```

@Schedule - Schedules a Runnable using a **ScheduledExecutionService**. Indifferent from **@ExecutorSvc** this does not require an **@Managed** also, but do work with **@Managed** if that is used to inject an instance of **Runnable** to be scheduled. **@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 instantiated). 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 now more 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 specify 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 container 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 Vaadin 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.

APSActivatorPlugin

Any implementing classes of this interface can be specified in

META-INF/services/se.natusoft.osgi.aps.tools.APSActivatorPlugin file, one per line. These are loaded by java.util.ServiceLoader. The implementation can be provided by another bundle which should then export the relevant packages which can then be imported in the using bundle.

The APSActivatorPlugin API looks like this:

```
public interface APSActivatorPlugin {

    interface ActivatorInteraction {
        void addManagedInstance(Object instance, Class forClass);
    }

    void analyseBundleClass(ActivatorInteraction activatorInteraction, Class bundleClass);
}
```

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

APSGlobalContextWrapper

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

```
Service providedService = APSGlobalContextWrapper.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/>.

APSConfigManager

This bundle listens for other bundles and checks for the following MANIFEST.MF entries:

- `APS-Config-Id` A unique id for a configuration to manage.
- `APS-Config-Schema` Points to a bundle relative JSON file containing an APS JSON schema, describing the configuration file. See `aps-core-lib` documentation for more information on the schema format.
- `APS-Config-Default-Resource` Points to a bundle relative JSON file following the schema and containing default configuration values.

For each bundle with an `APS-Config-Id` an instance of `APSConfig` is created loaded with current config from disk or cluster, or the default config file. If the config is new and has no previous configuration stored then the contents of the default configuration will be stored.

The created `APSConfig` is then published as an OSGi service and the following lookup property:

```
(apsConfigId=<config id>).
```

There are 2 ways to get configuration:

Since the configuration is published as an OSGi "service", a service tracker can be used to look it up. The `APSServiceTracker` has some special features that makes it easier and it can also be provided by `APSActivator`:

```
@OSGiService( additionalSearchCriteria = "(apsConfigId=my-config-id)", nonBlocking = true )
private APSServiceTracker<APSConfig> configTracker

@Managed
private BundleContext context

private APSConfig config

@Initializer
void init() {
    this.configTracker.onActiveServiceAvailable { APSConfig config, ServiceReference configRef ->
        this.config = this.context.getService(configRef);
        ...
    }

    this.configTracker.onActiveServiceLeaving { ServiceReference configRef, Class type ->
        this.config = null
        ...
    }
    ...
}
```

Note that the first argument which actually is the `APSConfig` instance you want, cannot be saved. It only exists within the callback block. Thereby we must allocate a new usage by using the OSGi `BundleContext` to get it. This should of course be released again when your code no longer needs the configuration object.

The other and easier way to get configuration is to let `APSActivator` give it to you:

```
private APSConfig config

@ConfigListener(apsConfigId = "my-config-id")
void config( APSConfig config ) {
    if (config != null) {
```



```

        this.config = config
        ...
    }
    else {
        this.config = null
    }
}

```

This is much easier, but actually does the same as the first example. The difference is that APSActivator does it for you. Note that the method will be called with a null value if the configuration is unpublished as an OSGi service! This will however only happen if the config is unpublished before aps-config-manager and the listening bundle shuts down. So basically your code must expect and handle a null, but cannot demand to always get a null on shutdown.

Note that in both these cases you wait for the configuration to become available. The following would also be possible and can seem simpler:

```

@OSGiService
private APSConfig config

```

But in this case you will be blocking the thread if it is accessed before it is available. If this was done in code called from bundle activator then there would be a problem! The other two ways are completely non blocking and reactive in that you get it when it is available.

Whatever the code is doing, it can start doing it when the config is available. If it needs to do something that isn't quick, then it should work in another thread since the config listener call made by APSActivator needs to return rather quickly. Here is a suggestion: `APSExecutor.submit { ... }`. This will then be submitted to a thread pool with as many threads as there are cores in the machine.

APSActivator can also inject an `ExecutionService` that is backed by a thread pool for use by the bundle only. `APSExecutor` provides one thread pool for all bundles to share and are intended for shorter jobs.

MapJSON

Configurations are JSON documents. `APSConfig` extends `Map<String, Object>` and can thus be used as a Map representation of the JSON data, but it also has `lookup(String key)` method that takes a structured key (implemented by `StructMap` from `aps-core-lib`) to lookup a value. There is also a `provide(String structPath, Object value)` method that is intended for configuration editors to use to update config values. An update of config value will update cluster and inform other nodes of the change. See documentation for `StructMap` in `aps-core-lib` for more information.

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 provides 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 *APSPFile*.

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

APSIException- 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

APSIException- on any failure.

}

APJSONLib

This is a library (exports all its packages and provides no service) for reading and writing JSON. It can also write a JavaBean object as JSON and take a JSON value or inputstream containing JSON and produce a JavaBean.

This basically provides a class representing each JSON type: JSONObject, JSONString, JSONNumber, JSONBoolean, JSONArray, JSONNull, and a JSONValue class that is the common base class for all the other. Each class knows how to read and write the JSON type it represents. Then there is a JavaToJSON and a JSONToJava class with static methods for converting back and forth. This mapping is very primitive. There has to be one to one between the JSON and the Java objects.

Changes

0.10.0

readJSON(...) in the **JSONValue** base class now throws JSONEOFException (extends IOException) on EOF. The reason for this is that internally it reads characters which cannot return -1 or any non JSON data valid char to represent EOF. Yes, it would be possible to replace *char* with *Character*, but that will have a greater effect on existing code using this lib. If an JSONEOFException comes and is not handled it is still very much more clear what happened than a NullPointerException would be!

APIs

Complete javadocs can be found at <http://apidoc.natusoft.se/APJSONLib/>.

```
public class JSONConversionException extends RuntimeException [se.natusoft.osgi.aps.json] {
```

This exception is thrown on failure to convert from JSON to Java or Java to JSON.

Almost all exceptions within the APS services and libraries extend either *APSEException* or *APSRuntimeException*. I decided to just extend RuntimeException here to avoid any other dependencies for this library since it can be useful outside of APS and can be used as any jar if not deployed in OSGi container.

```
public JSONConversionException(final String message)
```

Creates a new *JSONConversionException*.

Parameters

message- The exception message

```
public JSONConversionException(final String message, final Throwable cause)
```

Creates a new *JSONConversionException*.

Parameters

message- The exception message

cause- The cause of this exception.

```
}
```

```
public JSONArray()
```

Creates a new JSONArray for writing JSON output.

public JSONArray(JSONErrorHandler errorHandler)

Creates a new JSONArray for reading JSON input and writing JSON output.

Parameters

errorHandler- The error handler to use.

public void addValue(JSONValue value)

Adds a value to the array.

Parameters

value- The value to add.

}

public BeanInstance(Object modelInstance)

Creates a new ModelInstance.

Parameters

modelInstance- The model instance to wrap.

public Object getModelInstance()

Returns the test model instance held by this object.

public List<String> getSettableProperties()

Returns a list of settable properties.

public List<String> getGettableProperties()

Returns a list of gettable properties.

public void setProperty(String property, Object value) throws JSONConversionException

Sets a property

Parameters

property- The name of the property to set.

value- The value to set with.

Throws

JSONConversionException- on any failure to set the property.

public Object getProperty(String property) throws JSONConversionException

Returns the value of the specified property.

Returns

The property value.

Parameters

property- The property to return value of.

Throws

JSONConversionException- on failure (probably bad property name!).

public Class getPropertyType(String property) throws JSONConversionException

Returns the type of the specified property.

Returns

The class representing the property type.

Parameters

property- The property to get the type for.

Throws

JSONConversionException- if property does not exist.

}

**public static void read(InputStream jsonIn, APSHandler<APSResult<JSONValue>>
resultHandler)**

Reads any JSON object from the specified *InputStream*.

Parameters

jsonIn- The *InputStream* to read from.

resultHandler- The handler to call with result.

**public static void readToMap(InputStream jsonIn, APSHandler<APSResult<Map<String,
Object>>> resultHandler)**

Reads a JSON InputSteam and returns the JSON structure as a Map<String, Object>.

Parameters

jsonIn- The JSON stream to read.

resultHandler- The handler to receive the result.

**public static void resourceToMap(String resource, APSHandler<APSResult<Map<String,
Object>>> resultHandler)**

Reads a JSON classpath resource and returns the JSON structure as a Map<String, Object>.

Parameters

resource- The resource to read.

resultHandler- The handler to receive the result.

public static JSONValue read(InputStream jsonIn, JSONErrorHandler errorHandler)

Reads any JSON object from the specified *InputStream*.

Returns

A *JSONValue* subclass. Which depends on what was found on the stream.

Parameters

jsonIn- The *InputStream* to read from.

errorHandler- An implementation of this interface should be supplied by the user to handle any errors during JSON parsing.

Throws

APSIException- on any IO failures.

public static void write(OutputStream jsonOut, JSONValue value) throws APSIOException

Writes a *JSONValue* to an *OutputStream*. This will write compact output by default.

Parameters

jsonOut- The *OutputStream* to write to.

value- The value to write.

Throws

APSIException- on failure.

public static void write(OutputStream jsonOut, JSONValue value, boolean compact, APSHandler<APResult<Void>> resultHandler)

Writes a *JSONValue* to an *OutputStream*. This will write compact output by default.

Parameters

jsonOut- The *OutputStream* to write to.

value- The value to write.

resultHandler- handler for result. only success() or failure() is relevant.

public static void write(OutputStream jsonOut, JSONValue value, boolean compact) throws APSIOException

Writes a *JSONValue* to an *OutputStream*.

Parameters

jsonOut- The *OutputStream* to write to.

value- The value to write.

compact- If true the written JSON is made very compact and hard to read but produce less data.

Throws

APIOException- on IO problems.

public static byte[] jsonToBytes(JSONValue jsonValue) throws APIOException

Converts a JSONValue into bytes.

Returns

A byte array.

Parameters

jsonValue- The JSONValue to convert.

Throws

APIOException- on any IO failure.

public static JSONValue bytesToJson(byte[] bytes)

Converts a byte array into a JSONValue object. For this to work the byte array of course must contain valid JSON!

Parameters

bytes- The bytes to convert.

public static String jsonToString(JSONValue jsonValue) throws APIOException

Converts a JSONValue to a String of JSON.

Returns

A String of JSON.

Parameters

jsonValue- The json value to convert.

Throws

APIOException- on failure. Since the JSON is valid and we are writing to memory this is unlikely ...

public static JSONValue stringToJson(String jsonString) throws APIOException

Converts a String with JSON into a JSONValue.

Returns

Whatever JSON object the string contained, as a base JSONValue.

Parameters

jsonString- The JSON String to convert.

Throws

APIOException- on failure, like bad JSON in string.

public static Map<String, Object> jsonObjectToMap(JSONObject jsonObject)

This takes a JSONObject and returns a Map.

Returns

The converted Map.

Parameters

jsonObject- The JSONObject to convert to a Map.

public static JSONObject mapToJSONObject(Map<String, Object> map)

Converts a Map<String, Object> to a JSONObject.

Returns

A converted JSONObject.

Parameters

map- The Map to convert.

public static Map<String, Object> readJSONAsMap(InputStream jsonIn, JSONErrorHandler errorHandler)

For consistency. The same as doing JSON.jsonObjectToMap(InputStream, JSONErrorHandler).

Returns

A Map<String, Object> of JSON data.

Parameters

jsonIn- The input stream to read.

errorHandler- The error handler to use.

public static Map<String, Object> stringToMap(String json)

Converts from String to JSON to Map.

Returns

A Map representation of the JSON.

Parameters

json- The JSON String to convert.

public static String mapToString(Map<String, Object> map)

Converts from Map to JSONObject to String.

Returns

A String containing JSON.

Parameters

map- The Map to convert.

}

public class **CollectingErrorHandler** implements JSONErrorHandler [se.natusoft.osgi.aps.json] {

Utility implementation of JSONErrorHandler.

public CollectingErrorHandler(boolean printWarnings)

Parameters

printWarnings- If true warnings will be printed to stderr.

public boolean hasMessages()

Returns

true if there are any messages.

public String toString()

Returns

All messages as one string.

}

protected JSONValue()

Creates a new JSONValue.

protected JSONValue(JSONErrorHandler errorHandler)

Creates a new JSONValue

protected abstract void readJSON(char c, JSONReader reader) throws APSIOException

This will read the vale from an input stream.

Parameters

c- The first character already read from the input stream.

reader- The reader to read from.

Throws

APSIOException- on IO failure.

protected abstract void writeJSON(JSONWriter writer, boolean compact) throws APSIOException

This will write the data held by this JSON value in JSON format on the specified stream.

Parameters

writer- A JSONWriter instance to write with.

compact- If true write the JSON as compact as possible. false means readable, indented.

Throws

APIOException- On IO failure.

protected JSONErrorHandler getErrorHandler()

Returns

The user supplied error handler.

/*package*/

Reads and resolves what JSON type is the next in the input and returns it.

Returns

The read JSONValue.

Parameters

c- The first already read character.

reader- The reader to read from.

errorHandler- The user supplied error handler.

Throws

APIOException- on IOFailure.

protected void fail(String message, Throwable cause)

Fails the job.

Parameters

message- The failure message.

cause- An eventual cause of the failure. Can be null.

protected void fail(String message)

Fails the job.

Parameters

message- The failure message.

public void readJSON(InputStream is) throws APIOException

This will read the value from an input stream.

Parameters

is- The input stream to read from.

Throws

APSIOException- on IO failure.

public void writeJSON(OutputStream os) throws APSIOException

This writes JSON to the specified OutputStream.

Parameters

os- The outoutStream to write to.

Throws

APSIOException- on IO failure.

public void writeJSON(OutputStream os, boolean compact) throws APSIOException

This writes JSON to the specified OutputStream.

Parameters

os- The outoutStream to write to.

compact- If true write JSON as compact as possible. If false write it readable with indents.

Throws

APSIOException- on IO failure.

/*package*/

Method for creating a JSONString instance.

Parameters

errorHandler- The user error handler.

/*package*/

Method for creating a JSONNumber instance.

Parameters

errorHandler- The user error handler.

/*package*/

Method for creating a JSONNull instance.

Parameters

errorHandler- The user error handler.

/*package*/

Method for creating a JSONBoolean instance.

Parameters

errorHandler- The user error handler.

/*package*/

Method for creating a JSONArray instance.

Parameters

errorHandler- The user error handler.

/*package*/

Method for creating a JSONObject instance.

Parameters

errorHandler- The user error handler.

protected JSONReader(PushbackReader reader, JSONErrorHandler errorHandler)

Creates a new JSONReader instance.

Parameters

reader- The PushbackReader to read from.

errorHandler- The handler for errors.

protected char getChar() throws APSIOException

Returns the next character on the specified input stream, setting EOF state checkable with isEOF().

Throws

APSIOException- on IO problems.

protected static class **JSONWriter** [se.natusoft.osgi.aps.json] {

For subclasses to use in writeJSON(JSONWriter writer).

protected JSONWriter(Writer writer)

Creates a new JSONWriter instance.

Parameters

writer- The writer to write to.

protected void write(String json) throws APSIOException

Writes JSON output.

Parameters

json- The JSON output to write.

Throws

APIOException- on IO failure.

}

public JSONNumber(Number value)

Creates a new JSONNumber instance for writing JSON output.

Parameters

value- The numeric value.

public JSONNumber(JSONErrorHandler errorHandler)

Creates a new JSONNumber instance for reading JSON input or writing JSON output.

Parameters

errorHandler- The error handle to use.

public Number toNumber()

Returns the number as a Number.

public float toFloat()

Returns the number as a float value.

public int toInt()

Returns the number as an int value.

public long toLong()

Returns the number as a long value.

public short toShort()

Returns the number as a short value.

public byte toByte()

Returns the number as a byte value.

public String toString()

Returns

number as String.

public Object to(Class type)

Returns the number as a value of the type specified by the type parameter.

Parameters

type- The type of the returned number.

}

public static JSONObject convertObject(Object javaBean) throws JSONException

Converts a JavaBean object into a *JSONObject*.

Returns

A JSONObject containing all values from the JavaBean.

Parameters

javaBean- The JavaBean object to convert.

Throws

JSONException- on converting failure.

public static JSONObject convertObject(JSONObject jsonObject, Object javaBean) throws JSONException

Converts a JavaBean object into a *JSONObject*.

Returns

A JSONObject containing all values from the JavaBean.

Parameters

jsonObject- The jsonObject to convert the bean into or null for a new JSONObject.

javaBean- The JavaBean object to convert.

Throws

JSONException- on converting failure.

public static JSONValue convertValue(Object value)

Converts a value from a java value to a *JSONValue*.

Returns

The converted JSONValue.

Parameters

value- The java value to convert. It can be one of String, Number, Boolean, null, JavaBean, or an array of those.

}

public JSONNull()

Creates a new JSONNull instance for writing JSON output.

public JSONNull(JSONErrorHandler errorHandler)

Creates a new JSONNull instance for reading JSON input or writing JSON output.

Parameters

errorHandler- The error handler to use.

public String toString()*Returns*

as String.

}

public JSONObject()

Creates a JSONObject instance for writing JSON output.

public JSONObject(JSONErrorHandler errorHandler)

Creates a new JSONObject instance for reading JSON input or writing JSON output.

Parameters

errorHandler- The error handler to use.

public Set<JSONString> getValueNames()

Returns the names of the available properties.

public JSONValue getValue(JSONString name)

Returns the named property.

Parameters

name- The name of the property to get.

public JSONValue getValue(String name)

Returns the named property.

Parameters

name- The name of the property to get.

public void setValue(JSONString name, JSONValue value)

Adds a value to this JSONObject instance.

Parameters

name- The name of the value.

value- The value.

public void setValue(String name, String value)

Adds a string value.

Parameters

name- The name of the value.

value- The value.

public void setValue(String name, Number value)

Adds a numeric value.

Parameters

name- The name of the value.

value- The value.

public void setValue(String name, boolean value)

Adds a boolean vlaue.

Parameters

name- The name of the value.

value- The value.

public void fromMap(Map<String, Object> map)

populates this JSONObject from the specified Map.

Parameters

map- The Map to import.

public Map<String, Object> toMap()

Returns the JSONObject as a Map.

public void setValue(String name, JSONValue value)

Adds a property to this JSONObject instance.

Parameters

name- The name of the property.

value- The property value.

}

public JSONString(String value)

Creates a new JSONString for writing JSON output.

Parameters

value- The value of this JSONString.

public JSONString(JSONErrorHandler errorHandler)

Creates a new JSONString for reading JSON input and writing JSON output.

Parameters

errorHandler- The error handler to use.

}

public class JSOToJava [se.natusoft.osgi.aps.json] {

Creates a JavaBean instance and copies data from a JSON value to it.

The following mappings are made in addition to the expected ones:

- *JSONArray* only maps to an array property.
- Date properties in bean are mapped from *JSONString* "yyyy-MM-dd HH:mm:ss".
- Enum properties in bean are mapped from *JSONString* which have to contain enum constant name.

public static <T> T convert(InputStream jsonStream, Class<T> javaClass) throws APSIOException, JSONConversionException

Returns an instance of a java class populated with data from a json object value read from a stream.

Returns

A populated instance of javaClass.

Parameters

jsonStream- The stream to read from.

javaClass- The java class to instantiate and populate.

Throws

APSIOException- on IO failures.

JSONConversionException- On JSON to Java failures.

public static <T> T convert(String json, Class<T> javaClass) throws APSIOException, JSONConversionException

Returns an instance of a java class populated with data from a json object value read from a String containing JSON.

Returns

A populated instance of javaClass.

Parameters

json- The String to read from.

javaClass- The java class to instantiate and populate.

Throws

APSIOException- on IO failures.

JSONConversionException- On JSON to Java failures.

public static <T> T convert(JSONValue json, Class<T> javaClass) throws JSONConversionException

Returns an instance of java class populated with data from json.

Returns

A converted Java object.

Parameters

json- The json to convert to java.

javaClass- The class of the java object to convert to.

Throws

JSONConversionException- On failure to convert.

}

public class SystemOutErrorHandler implements JSONErrorHandler [se.natusoft.osgi.aps.json] {

A simple implementation of *JSONErrorHandler* that simply displays messages on System.out and throws a *RuntimeException* on fail. This is used by the tests. In a non test case another implementation is probably preferred.

}

public JSONBoolean(boolean value)

Creates a new JSONBoolean instance for writing JSON output.

Parameters

value- The value for this boolean.

public JSONBoolean(JSONErrorHandler errorHandler)

Creates a new JSONBoolean instance for reading JSON input or writing JSON output.

Parameters

errorHandler- The error handler to use.

public void setBooleanValue(boolean value)

Sets the value of this boolean.

Parameters

value- The value to set.

public boolean getAsBoolean()

Returns the value of this boolean.

public String toString()

Returns the value of this boolean as a String.

public Boolean toBoolean()

Returns

this JSONBoolean as a Java boolean.

}

}

public interface JSONErrorHandler [se.natusoft.osgi.aps.json] {

This is called on warnings or failures.

@author Tommy Svensson

void warning(String message)

Warns about something.

Parameters

message- The warning message.

void fail(String message, Throwable cause) throws RuntimeException

Indicate failure.

Parameters

message- The failure message.

cause- The cause of the failure. Can be null!

Throws

RuntimeException- This method must throw a *RuntimeException*.

}

aps-core-lib

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 `APSVValidationException`.

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 by defining a map with only one entry. Example:

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

In this case `nameAddress` can contain any number of entries as long as each entry have a key containing 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.

APSJPAService

This provides JPA to services and applications. It has a slightly more OSGi friendly API than the `org.osgi.service.jpa.EntityManagerFactoryBuilder`. The `APSOpenJPAProvider` however returns an `APSJPAService` instance that also implements `EntityManagerFactoryBuilder`. For some reason I haven't figured out yet, it cannot be registered as a service with the `EntityManagerFactoryBuilder` interface! The bundle fails to deploy if that is done.

The provided service is using OpenJPA. The service works partly as an extender inspecting deployed bundles for a `META-INF/persistence.xml` file. When found this is read and some setup is done already there. The `persistenceUnitName` from the `persistence.xml` file is used to connect the client later with its configuration. When a JPA using bundle is shut down its JPA setup is automatically cleaned.

Here is an example of usage:

```
private APSJPAEntityManagerProvider emp = null;
...
private APSJPAEntityManagerProvider getEMP() {
    if (this.emp == null || !this.emp.isValid()) {
        DataSourceDef dsDef = this.dataSourceDefService.lookupByName("MyDS");
        if (dsDef == null) {
            throw new SomeException("Could not find an 'MyDs' in 'persistence/datasources'
configuration!");
        }
        Map<String, String> props = new HashMap<String, String>();
        props.put("javax.persistence.jdbc.user", dsDef.getConnectionUserName());
        props.put("javax.persistence.jdbc.password", dsDef.getConnectionPassword());
        props.put("javax.persistence.jdbc.url", dsDef.getConnectionURL());
        props.put("javax.persistence.jdbc.driver", dsDef.getConnectionDriveName());
        this.emp = this.jpaService.initialize(this.bundleContext, "myPersistenceUnitName",
props);
    }
    return this.emp;
}
...
EntityManager em = getEMP().createEntityManager();
em.getTransaction().begin();

try {
    RoleEntity role = new RoleEntity(id);
    role.setDescription(description);
    em.persist(role);
    em.getTransaction().commit();
}
catch (RuntimeException re) {
    em.getTransaction().rollback();
    throw re;
}
finally {
    em.close();
}
```

This code example handles the `APSJPAService` having been restarted or redeployed. When `emp.isValid()` returns false then all you need to do is to call `jpaService.initialize(...)` again. The rest is just POJPA (Plain Old JPA :-)).

APIs

```
public interface APSJPAService [se.natusoft.osgi.aps.api.data.jpa.service] {
```

This service allows an JPA *EntityManager* to be gotten for a persistent unit name.

So why is this done this way ? Why is not an *EntityManagerFactory* returned?

The answer to that is that the *EntityManagerFactory* is internal to the service who is responsible for creating it and for closing it at sometime (stopping of bundle). The client only needs an *EntityManager* for which the client is responsible after its creation.

The creation of the *EntityManagerFactory* is delayed until the call to *initialize(...)*. Creating the EMF along with the persistence provider at persistence bundle discovery would limit database connection properties to the persistence.xml file which is less than optimal to put it mildly. This way a client can make use of the *APSDDataSourceDefService* to get the JDBC properties which it can pass along to this service.

The default provider implementation of this service uses OpenJPA which provides its own connection pooling.

APSJPAEntityManagerProvider initialize(BundleContext bundleContext, String persistenceUnitName, Map<String, String> props) throws APSResourceNotFoundException

Initializes and returns a provider from the specified properties.

Returns

A configured EntityManager.

Parameters

bundleContext- The context of the client bundle. It is used to locate its persistence provider.

persistenceUnitName- The name of the persistent unit defined in persistence.xml.

props- Custom properties to configure database, etc.

boolean isValid()

Returns true if this instance is valid. If not call *APSJPAService.initialize(...)* again to get a new instance. It will be invalid if the *APSJPAService* provider have been restarted.

EntityManager createEntityManager()

Creates a new *EntityManager*. You are responsible for closing it!

Please note that the *EntityManager* caches all referenced entities. If you keep and reuse it for a longer time it can use more memory. For example at http://docs.jboss.org/ejb3/app-server/tutorial/extended_pc/extended.html it says that "Usually, an *EntityManager* in JBoss EJB 3.0 lives and dies within a JTA transaction". This indicates how long-lived the *EntityManager* should preferably be.

Returns

A configured EntityManager.

EntityManagerFactory getEntityManagerFactory()

Returns the underlying *EntityManagerFactory*. This will return null if *isValid()* return false!

Be very careful what you do with this! It is managed by this service!

}

APS RabbitMQ Message Service Provider

This service provides an implementation of `APSMessagingService` using [RabbitMQ](#).

Note: This implementation does not support `contentType` in the API. When sending messages the `contentType` will be ignored, and when messages are received the `contentType` will always be "UNKNOWN".

A good suggestion is to always use JSON or XML as content.

APSMessagingService API

[Javadoc](#)

aps-vertex-event-bus-messaging-provider

This publishes 3 services: `MessagePublisher`, `MessageSender`, and `MessageSubscriber`.

Each of these use Vertx EventBus under the surface. These messages are not persistent and are sent within a Vertx cluster.

The difference between publisher and sender is that publisher sends to everyone listening on the destination, while sender only sends to one. Vertx uses round robin for sender to spread out to nodes in the cluster.

The `aps-vertex-provider` also publishes the EventBus instance as a service, which is what this implementation is using. The EventBus can of course be used directly instead of this service. This service uses the official APS messaging API and thus all kinds of messaging can be done in the same way. It is also in general a good idea encapsulate the real implementation so that it is easy to change to something else if needed/wanted in the future. This service provides that.

For more information on Vertx see: <http://vertx.io/docs/vertx-core/groovy/>

Lookup

This properties for all 3 services contains:

```
aps-protocol-name:    vertx-eventbus
service-category:     network
service-function:     messaging
messaging-persistent: false
messaging-clustered:  true
service-provider:     aps-vertex-event-bus-messaging-provider:publisher/sender/subscriber
```

Web Manager

Components

The components are mostly based on React-Bootstrap components. The APS components wrap other components to provide components that connect to a common bus. All relevant component events are sent as messages on the bus. No code has to attach listeners to specific components. Any code that wants to react to a component being triggered does so by joining the bus and listen to messages. So the code reacting on messages really has no direct association with any component, they just react to messages.

All components has a `listenTo` and a `sendTo` property which points out an address to send or listen to. Most components only send messages.

This separation of component rendering code and event handler code makes it rather easy to render components after a JSON specification. That is what WebManager does. It listens for a JSON GUI spec on the bus, and when it sees such, it renders it. When a user interacts with the rendered GUI messages will be sent on the bus.

APSEventBus

On the frontend there is a local `APSEventBus` that is used to send and receive messages. This is basically a wrapper. It does not do anything other than pass on to 'EventBusRouter's. There are 2 'EventBusRouter's: `APSLocalEventBusRouter` and `APSVertxEventBusRouter`. These are both added to `APSEventBus`.

APSLocalEventBusRouter

This looks at the header of a message and if *routing* contains *client* then it passes the message to all registered local subscribers. This never goes out on the network, it only works internally in the client. This is used to communicate between different components locally.

APSVertxEventBusRouter

This looks at the header of a message and if *routing* contains *backend* it does a send on the Vert.x client event bus bridge. A send does a round robin to listeners of specified address. If *routing* contains *cluster* it does a publish on the Vert.x client event bus bridge. This goes to all listeners of the address in the whole cluster, including other clients.

Messaging

A note: *Vert.x uses Hazelcast by default to handle the cluster. Hazelcast by default manages a cluster on the same subnet only, by multicast. Hazelcast can of course be configured to create larger clusters. But that is a Hazelcast thing, not APS nor Vert.x.*

Addresses

Each message must be addressed! An address is just a string. It can be anything. If you send to an address it will be received if there is something also listening to exactly the same address. See an address as a group name as is common in other messaging solutions.

As said above when a send is done on the Vert.x event bus it does a round robin between each listener of the "sent to" address. This to support load balancing.

Address Strategy

Note: (app) refers to a specific application as a wildcard, and (UUID) refers to a generated UUID value.

From **client perspective**:

- Client

- "aps:(app):client:(UUID)"
 - Network: listen
 - Local: listen, send
- Backend
 - "aps:(app):backend"
 - Network: send
- Cluster
 - "aps:(app):all" : Every backend and client of (app).
 - Network: listen, send
 - "aps:(app):all:backend" : Every backend of (app).
 - Network: send
 - "aps:(app):all:client" : Every client of (app).
 - Network: listen, send

Note 1: Only clients have a unique address. They are the only ones that needs a unique address due to being the only unique thing.

Note 2: Vert.x does a round robin on "send" to same address. Only backend messages are delivered with a send. All cluster messages are delived with a "publish" and will always go to every subscriber.

Note 3: Both on client side and on backend, code is not interacting with the Vert.x bus, but local busses with message routers. So what messages are sent on the Vert.x bus and which message method is used is a routing question. Senders and receivers don't need to care.

Routing Strategy

My first though was to hide routing as much as possible, but that creates a lot of limitations.

Components have indirect routing information passed to them by the creator of the component, and in general all routing is handled by the `APSCoComponent` base class for the APS components. Individual components should ignore routing completely. They just send and possibly receive messages not caring about anything else than the message.

Some components need to talk to each other locally, and some need to reach a backend. It is up to the code that creates the components to determine that by supplying a routing property of:

```
{
  outgoing: "client/backend/all/all:backend/all:client/none",
  incoming: "client/all/all:client/none"
}
```

Both outgoing and incomming can have more than one route. Routes are comma separated within the string. No spaces.

Do note the "none" at the end! Its more relevant for incoming than outgoing. It means what the word says. If a component does not react to any incoming messages, then there is no point in listening to incoming messages and the messaging code does not need to make an unceccesary call to a component that will do nothing with the provided message.

Messages

When a message is sent, there is no from, there is only a to. My goal is that all code should only react to messages, and a reaction can possibly be a new message. But there should not be any specific reply message. A message might be reacted to by a client that updates something in a GUI, or it might be reacted to in the backend. Whatever sends the message sholdn't care. Where messages end upp is a routing question.

The `content` part can be missing or empty depending on `type`.

In the following specification any entry starting with '?' is optional.

General

```

{
  "aps": { // To be moved to header instead ...
    "origin": "(address)",
    "app": "(app)",
    "type": "(message type)",
    "identity": {
      "userId": "(id)",
      "userName": "(name of user)",
      "auth": {
        "type": "(authentication type)",
        (authentication specific data)
      }
    }
  },
  content: { // Will probably go away when "aps" is moved to header.
    ...message type specific data
  }
}

```

--> Avail

Client tells a backend that it exists and are ready for a GUI JSON document.

This is not a requirement, components can be created and a gui built with client code like any other React GUI. There are 2 components that works together and use this message: `APSPage` and `APSWebManager`. They will as default inform the backend that they are up and running and the backend will send a JSON document with components to create. `APSWebManager` can however also be used with a property that tells it to not send the message, and only act as container, but create and supplying a common event bus to all children.

```

{
  "aps": {
    "origin": "(address)"
    "app": "(app)",
    "type": "avail"
  }
}

```

<-- Gui

Client receives a gui, most probably from backend, but can really be send from anywhere including client itself.

```

{ "aps": { "origin": "(address)", "app": "(app)", "type": "gui" }, "content": { "id": "(comp id)", "name": "(comp name)", ?"group": "(name of group component belongs to if any.)", "type": "(comp type)", ?"class": "(class class ...)", ?"disabled": true/false, ?"enabled": "groupNotEmpty:(group)/namedComponentsNotEmpty:(names)" "collectGroups": "(space separated list of groups to listen to and collect data from.)", "(type specific)": "(...)", ..., "children": ... } }

```

--> Create

```

{
  "aps": {
    "origin": 'address'
    "app": 'app',
    "type": "create"
  }
  "content": {
  }
}

```

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-all-2.4.15](#)
- [vertx-core-3.4.2](#)
- [vertx-auth-common-3.5.3](#)
- [vertx-lang-groovy-3.5.3](#)
- [vertx-web-3.5.3](#)
- [vertx-hazelcast-3.5.3](#)
- [hazelcast-3.6.3](#)
- [jackson-core-2.7.4](#)
- [jackson-annotations-2.7.0](#)
- [jackson-databind-2.7.4](#)
- [netty-codec-4.1.6.Final](#)
- [netty-codec-http-4.1.6.Final](#)
- [netty-codec-http2-4.1.6.Final](#)
- [netty-codec-dns-4.1.6.Final](#)
- [netty-codec-socks-4.1.6.Final](#)
- [netty-handler-4.1.6.Final](#)
- [netty-handler-proxy-4.1.6.Final](#)
- [netty-resolver-4.1.6.Final](#)
- [netty-resolver-dns-4.1.6.Final](#)
- [netty-buffer-4.1.6.Final](#)
- [netty-common-4.1.6.Final](#)
- [netty-transport-4.1.6.Final](#)
- [geronimo-jta_1.1_spec-1.1.1](#)
- [commons-net-3.5](#)
- [openjpa-all-2.2.0](#)

Eclipse Public License -v 1.0 version 1.0

The following third party products are using this license:

- [javax.persistence-2.0.0](#)

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.

Eclipse Public License -v 1.0 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.

"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.

2. 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,
- 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 use the Program in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system;
- 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 third party; and
- 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.

3. 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.

4. 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, 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

5. 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 WARRANTY

6. 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 W

7. 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. 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) infringes its patent rights, it shall be deemed to have irrevocably and exclusively assigned to the copyright owner of the Program all of its right, title and interest in any patent litigation (including any counterclaim or cross-claim) brought by Recipient against any entity, and shall be deemed to have irrevocably and exclusively assigned to the copyright owner of the Program all of its right, title and interest in any patent litigation (including any counterclaim or cross-claim) brought by Recipient against any entity. 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 being notified of such failure. 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 may modify the Agreement to make it compatible with the laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement mo

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.