Application Platform Services

User Guide

1.0.0

Tommy Svensson

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 wont fit all. It can be seen as osgi-ee-light-and-easy. The services are of platform type: configuration, database, JPA, etc, with companion web applications for administration.

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

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

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

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

Features

Current

- A configuration service that works with annotated configuration models where each config value can be described/documented. The configuration model can be structured with sub models that there can be one or many of. Each top level configuration model registered with the configuration service will be available for publishing in the admin web. The configuration service also supports different configuration environments and allows for configuration values to be different for different configuration environments, but doesn´t require them to be.
- Synchronization of configurations across servers. There is currently 2 implementations for this, one that syncs using APSGroups service, one that syncs using RabbitMQ, and one that syncs via Hazelcast.
- 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.
- A data source service. Only provides connection information, no pooling (OpenJPA provides its own pooling)!
- External protocol extender that allows more or less any OSGi service to be called remotely using any deployed protocol service and transport. Currently provides JSONRPC 1.0 & 2.0, JSONHTTP, and JSONREST protocols, and an http transport. Protocols have a defined service API whose implementations can just be dropped in to make them available. Transport providers can make use of any deployed protocol. The APSExternalProtocolService now provides support for REST services where there is a method for post, put, get,and delete, and the http transport makes use of this in conjunction with any protocol that indicates it can support REST like JSONREST.
- A group service that can send data to each member over transport safe multicast.
- A service discovery service using the group service.
- A session service (not http!). This is used by apsadminweb to keep a session among several different administration web applications.
- An administration web service to which administration web applications can register themselves with an url and thus be available in the .../apsadminweb admin gui.
- A user service. Provides basic user management including roles/groups. Is accompanied with a admin GUI (plugnis into apsadminweb) for administration of users. (org.osgi.service.useradmin.UserAdmin felt uncomplete. It did not provide what I wanted).
- A user authentication service. This does nothing more that authenticating a user and have a really

simple API. APS provides an implementation that makes use of the user service, but it is easy to make another implementation that authenticates against an LDAP for example or something else. The Admin web applications uses the authentication service for authenticating admin users.
- 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.

Planned

- An implementation of the standard OSGi LogService since not all servers provide one.
- A log veiwer web application supporting requlars expression filters on log information and a live log view. This is waiting on Vaadin 7.1 which will support server push. Another alternative is to go pure GWT and use Errai for this, but I rather continue with Vaadin having all admin webs looking and feeling the same.
- Anything else relevant I come up with and consider fun to do :-).

Ideas

- A JCR (Java Content Repository) service and a content publishing GUI (following the general APS ambition - reasonable functionality and flexibility, ease of use. Will fit many, but not everyone).
- Support for being able to redeploy a web application and services live without loosing session nor user transactions. With OSGi it should be teoretically possible. For a limited number of redeployments at least. It is very easy to run into the "perm gen space" problem, but according to Frank Kieviet (Classloader leaks: The dreaded permgen space) it is caused by bad code and can be avoided.

What is new in

1.0.0

- Bug fix in APSConfigService that was forced to make it non backwards compatible to fix. Sorry for that! Using the APSConfigService work exactly as before, but editing config have changed. **The big catch however is that the keys in the configuration files have changed and thus old saved configurations no longer work!** I had no choice. The old keys where part of the problem. I admit that I did something very stupid in the first version and that I should have known better, and in the end I had no other choice than to fix it, which came as no surprice!
- Added Hazelcast support with APS Hazelcast configuration service.
-

0.10.0

Added syncrhonization services and made config synchronizable.

0.9.2

- Small bug fixes.
- APSActivator has been added to aps-tools-lib and can be used as bundle activator. It uses annotations to register services and inject tracked services and other things.
- A service can now be registered with an *aps-externalizable* property with value *true* to be made externally available by aps-external-protocol-extender.

0.9.1

- Now have full REST support in aps-external-protocol-extender and aps-ext-protocol-http-transport-provider.
- Documentation have been cleaned up a bit.

Requirements

The administration web application(s) are currently WABs and thus require a server supporting WAB

deployments. I have developed/tested this on Glassfish and Virgo. I am however considering seeing if it is possible to also support both Glassfish and JBoss JEE WAR to OSGi bridges. They are unfortunately very server specific since there are no such standard. Other than that all services are basic OSGi services and should theoretically run in any R4 compatible OSGi server.

Pre Setup

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

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

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

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

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

Please note that the /apsadminweb by default require no login! This so that *"Configurations tab, Configurations/persistence/datasources"* can be used to setup a datasource called "APSSimpleUserServiceDS" needed by APSSimpleUserService. If you use the provided APSAuthService implementation that uses APSSimpleUserService then you need to configure this datasource before APSSimpleUserService can be used. See the documentation for APSSimpleUserService further down in this document for more information on the datasource configuration. After that is setup go to *"Configurations tab, Configurations/aps/adminweb"* and enable the "requireauthentication" config. After having enabled this and saved, do a browser refresh and then provide userid and password when prompted.

Javadoc

The complete javadoc for all services can be found at http://apidoc.natusoft.se/APS.

APSFilesystemService

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

The APSConfigService uses the APSFilesystemService to store its configurations.

Setup

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

The service

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

The javadoc for the APSFilesystemService.

The APIs for this service

public *interface* **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

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* **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

OutputStream createOutputStream() throws IOException

Creates a new *OutputStream* to this file.

Throws

IOException

Reader createReader() throws IOException

Creates a new *Reader* to this file.

Throws

IOException

Writer createWriter() throws IOException

Creates a new *Writer* to this file.

Throws

IOException

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 isDirectory()

See

java.io.File.isDirectory()

boolean isFile()

See

java.io.File.isFile()

boolean isHidden()

See

java.io.File.isHidden()

long lastModified()

See

java.io.File.lastModified()

long length()

See

java.io.File.length()

boolean createNewFile() throws IOException

See

java.io.File.createNewFile()

boolean delete()

See

java.io.File.delete()

void deleteOnExit()

See

java.io.File.deleteOnExit()

String toString()

Returns a string representation of this *APSFile*.

File toFile()

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

Returns

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

}

---

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

This represents an *APSFilesystemService* filesystem.

APSDirectory getDirectory(String path) throws IOException

Returns a folder at the specified path.

Parameters

path- The path of the folder to get.

Throws

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

APSFile getFile(String path)

Returns the file or folder of the specified path.

Parameters

path- The path of the file.

APSDirectory getRootDirectory()

Returns the root directory.

}

---

public *interface* **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;

if (fss.hasFilesystem("my.file.system")) {
    fs = fss.getFilsystem("my.file.system");
}
else {
    fs = fss.createFilesystem("my.file.system");
```

```
    }
```

APSFilesystem createFilesystem(String owner) throws IOException

Creates a new filesystem for use by an application or service. Where on disk this filesystem resides is irrelevant. It is accessed using the "owner", and will exist until it is removed.

Parameters

owner- The owner of the filesystem or rather a unique identifier of it. Consider using application or service package.

Throws

IOException- on any failure. An already existing filesystem for the "owner" will cause this exception.

boolean hasFilesystem(String owner)

Returns true if the specified owner has a filesystem.

Parameters

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

APSFilesystem getFilesystem(String owner) throws IOException

Returns the filesystem for the specified owner.

Parameters

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

Throws

IOException- on any failure.

void deleteFilesystem(String owner) throws IOException

Removes the filesystem and all files in it.

Parameters

owner- The owner of the filesystem to delete.

Throws

IOException- on any failure.

}

APSPlatformService

This is a trivial little service that just returns meta data about the specific platform installation.

The returned information is configured in the */apsadminweb*.

APIs

public *class* **PlatformDescription** [se.natusoft.osgi.aps.api.core.platform.model] {

This model provides information about a platform installation.

public PlatformDescription()

Creates a new PlatformDescription.

public PlatformDescription(String identifier, String type, String description)

Creates a new PlatformDescription.

Parameters

identifier- An identifying name for the platform.

type- The type of the platform, for example "Development", "SystemTest".

description- A short description of the platform instance.

public String getIdentifier()

Returns the platform identifier.

public String getType()

Returns the type of the platform.

public String getDescription()

Returns the description of the platform.

}

---

public *interface* **APSPlatformService** [se.natusoft.osgi.aps.api.core.platform.service] {

Provides information about the platform instance.

public PlatformDescription getPlatformDescription()

Returns a description of the platform instance / installation.

}

APSJSONLib

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/APSJSONLib/.

public *class* **JSON** [se.natusoft.osgi.aps.json] {

This is the official API for reading and writing JSON values.

public static JSONValue read(InputStream jsonIn, JSONErrorHandler errorHandler) throws IOException

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

IOException- 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

APSIOException- on failure.

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

APSIOException- on IO problems.

}

---

public JSONArrayProvider()

Creates a new JSONArray for wrinting JSON output.

public JSONArrayProvider(JSONErrorHandler errorHandler)

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

Parameters

errorHandler- The error handler to use.

public void addValue(JSONValueProvider value)

Adds a value to the array.

Parameters

value- The value to add.

public List<se.natusoft.osgi.aps.api.misc.json.model.JSONValue> getAsList()

Returns the array values as a List.

public <T extends se.natusoft.osgi.aps.api.misc.json.model.JSONValue> List<T> getAsList(Class<T> type)

Returns the array values as a list of a specific type.

Returns

A list of specified type if type is the same as in the list.

Parameters

type- The class of the type to return values as a list of.

<T>- One of the JSONValue subclasses.

}

---

public JSONBooleanProvider(boolean value)

Creates a new JSONBoolean instance for writing JSON output.

Parameters

value- The value for this boolean.

public JSONBooleanProvider(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 JSONNullProvider()

Creates a new JSONNull instance for writing JSON output.

public JSONNullProvider(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 JSONNumberProvider(Number value)

Creates a new JSONNumber instance for writing JSON output.

Parameters

value- The numeric value.

public JSONNumberProvider(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 JSONObjectProvider()

Creates a JSONObject instance for writing JSON output.

public JSONObjectProvider(JSONErrorHandler errorHandler)

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

Parameters

errorHandler- The error handler to use.

public void addValue(se.natusoft.osgi.aps.api.misc.json.model.JSONString name, JSONValueProvider value)

Adds a property to this JSONObject instance.

Parameters

name- The name of the property.

value- The property value.

public void addValue(String name, se.natusoft.osgi.aps.api.misc.json.model.JSONValue value)

Adds a property to this JSONObject instance.

Parameters

name- The name of the property.

value- The property value.

}

---

public JSONStringProvider(String value)

Creates a new JSONString for writing JSON output.

Parameters

value- The value of this JSONString.

public JSONStringProvider(JSONErrorHandler errorHandler)

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

Parameters

errorHandler- The error handler to use.

}

---

protected JSONValueProvider()

Creates a new JSONValue.

protected JSONValueProvider(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

APSIOException- 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

APSIOException- 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 APSIOException

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

APSIOException- on IO failure.

}

---

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 JSONConvertionException

Sets a property

Parameters

property- The name of the property to set.

value- The value to set with.

Throws

JSONConvertionException- on any failure to set the property.

public Object getProperty(String property) throws JSONConvertionException

Returns the value of the specified property.

Returns

The property value.

Parameters

property- The property to return value of.

Throws

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

public Class getPropertyType(String property) throws JSONConvertionException

Returns the type of the specified property.

Returns

The class representing the property type.

Parameters

property- The property to get the type for.

Throws

JSONConvertionException- if property does not exist.

}

---

public static JSONObjectProvider convertObject(Object javaBean) throws JSONConvertionException

Converts a JavaBean object into a *JSONObject*.

Returns

A JSONObject containing all values from the JavaBean.

Parameters

javaBean- The JavaBean object to convert.

Throws

JSONConvertionException- on converting failure.

public static JSONObjectProvider convertObject(JSONObjectProvider jsonObject, Object javaBean) throws JSONConvertionException

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

JSONConvertionException- on converting failure.

public static JSONValueProvider 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 *class* **JSONConvertionException** extends RuntimeException [se.natusoft.osgi.aps.json.tools] {

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 *APSException* 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 JSONConvertionException(final String message)

Creates a new *JSONConvertionException*.

Parameters

message- The exception message

public JSONConvertionException(final String message, final Throwable cause)

Creates a new *JSONConvertionException*.

Parameters

message- The exception message

cause- The cause of this exception.

}

---

public *class* **JSONMapConv** [se.natusoft.osgi.aps.json.tools] {

This converts between a Java Map and JSON. Do note that this of course uses this library to read and write JSON, but this specific public API only deals with Java and JSON as String or on/in a stream. p/ This class becomes more useful when used from Groovy since the latter provides much nicer usage of data in Maps. Yes, I know about JSONSlurper and JSONBuilder in Groovy. Those however does not work with @CompileStatic. Maps does.

public static Map<String, Object> jsonObjectToMap(String json) throws APSIOException

This takes a String containing a JSON object and returns it as a Map.

Parameters

json- The JSON content to convert to a Map.

Throws

APSIOException- on failure.

public static Map<String, Object> jsonObjectToMap(se.natusoft.osgi.aps.api.misc.json.model.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 String mapToJSONObjectString(Map<String, Object> map) throws APSIOException

This takes a Map (as created by jsonObjectToMap(...)) and returns a JSON String.

Parameters

map- The Map to convert to JSON.

Throws

APSIOException- on I/O failures.

public static se.natusoft.osgi.aps.api.misc.json.model.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 *class* **JSONToJava** [se.natusoft.osgi.aps.json.tools] {

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, JSONConvertionException

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.

JSONConvertionException- On JSON to Java failures.

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

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.

JSONConvertionException- On JSON to Java failures.

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

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

JSONConvertionException- On failure to convert.

}

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

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.

}

APSToolsLib

This is a library of utilities including a service tracker that is far better than the default one, including exception rather than null response on timeout, timeout specification, getting a proxied service implementation that automatically uses the tracker, allocating a service, calling it, and deallocating it again. This makes it trivially easy to handle a service being restarted or redeployed. It also includes a logger utility that will lookup the standard log service and log to that if found, otherwise just log to stdout.

This bundle provides no services. It just makes all its packages public. Every bundle included in APS makes use of APSToolsLib so it must be deployed for things to work.

Please note that this bundle has no dependencies! That is, it can be used as is without requireing any other APS bundle. It however requires APSOSGiTestTools to build, but that is only a test dependency.

APSServiceTracker

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

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

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

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

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

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

On bundle stop you should do:

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

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

Services and active service

The tracker tracks all instances of the service being tracked. It however have the notion of an active service. The active service is the service instance that will be returned by allocateService() (which is internally used by all other access methods also). On startup the active service will be the first service instance received. It will keep tracking other instances comming in, but as long as the active

service does not go away it will be the one used. If the active service goes away then the the one that is at the beginning of the list of the other tracked instances will become active. If that list is empty there will be no active, which will trigger a wait for a service to become available again if allocateService() is called.

Providing a logger

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

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

When available the tracker will log to this.

Tracker as a wrapped service

The tracker can be used as a wrapped service:

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

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

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

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

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

To get a service instance you do:

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

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

When done with the service do:

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

Accessing a service by tracker callback

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

onServiceAvailable

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

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

onServiceLeaving

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

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

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

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

onActiveServiceAvailable

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

onActiveServiceLeaving

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

withService

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

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

```
tracker.withService(new WithService<Service>() {
    @Override
    public void withService(
        Service service,
        Object... args
    ) throws Exception {
        // do something here.
    }
}, arg1, arg2);
```

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

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

```
                Service service
            ) throws Exception {
                // do something here
            }
        });
```

withServiceIfAvailable

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

withAllAvailableServices

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

onTimeout (since 0.9.3)

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

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

APSLogger

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

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

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

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

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

APSActivator

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

This is actually a rather trivial class that just scans the bundle for classes and inspects all classes for annotations and act on them. Most methods are protected making it easy to subclass this class and expand on its functionality.

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

The following annotations are available:

@OSGiServiceProvider - This should be specified on a class that implements a service interface

and should be registered as an OSGi service. *Please note* that the first declared implemented interface is used as service interface unless you specify serviceAPIs={Svc.class, ...}.

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

public @interface OSGiServiceInstance {

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

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

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

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

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

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

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

    /**
     * If true this service will be started in a separate thread.
     * This means the bundle start will continue in parallel and
     * that any failures in startup will be logged, but will
     * not stop the bundle from being started. If this is true
     * it wins over required service dependencies of the service
     * class. Specifying this as true allows you to do things that
     * cannot be done in a bunde activator start method, like
     * calling a service tracked by APSServiceTracker, without
     * causing a deadlock.
     */
    boolean threadStart() default false;
}
```

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

@APSExternalizabe, **@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 APSNoServiceAvailableException (runtime) if no service has become available before the specified timeout. It is also possible to have APSServiceTracker as field type in which case the underlying configured tracker will be injected instead.

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

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

```
public @interface OSGiService {

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

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

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

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

    /**
     * If this is set to true and a proxied implementation of the service is injected rather than
    the tracker directly
     * then any call made to the proxy will be cached if the service is not available and then
    later run when the
     * service becomes available. This of course means that methods returning a value will always
    return null when
     * service is not currently available since the real call will be made in the future.
    Returning a Future instead
     * in this case does not work since 'Future's are blocking, and we try to avoid blocking here.
     *
     * __YOU HAVE TO BE VERY CAREFUL WHEN SETTING THIS TO TRUE! NO CALLS RETURNING A VALUE!__
     *
```

```
         * The point of this is to be non blocking. By default with a proxied implementation
        tracker.allocateService() will
         * be called, and this blocks waiting for the service to become available if it is not.
         *
         * @return true or false (default).
         */
        boolean nonBlocking() default false;
    }
```

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

```
    public @interface Managed {

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

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

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

```
    public @interface BundleStart {

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

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

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

Usage as BundleActivator

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

Other Usage

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

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

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

There are 2 support classes:

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

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

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.

APSContextWrapper

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

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

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

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

ID generators

There is one interface:

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

    /**
```

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

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

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

    }
```

that have 2 implementations:

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

Javadoc

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

APSWebTools

This is not an OSGi bundle! This is a plain jar containing utilities for web applications. Specifically APS administration web applications. This jar has to be included in each web application that wants to use it.

Among other things it provides support for being part of the APS administration web login (APSAdminWebLoginHandler). Since the APS administration web is built using Vaadin it has Vaadin support classes. APSVaadinOSGiApplication is a base class used by all APS administration webs.

APIs

The following are the APIs for a few selected classes. The complete javadoc for this library can be found at http://apidoc.natusoft.se/APSWebTools/.

---

public *class* **APSAdminWebLoginHandler** extends APSLoginHandler implements APSLoginHandler.HandlerInfo [se.natusoft.osgi.aps.tools.web] {

This is a login handler to use by any admin web registering with the *APSAdminWeb* to validate that there is a valid login available.

public APSAdminWebLoginHandler(BundleContext context)

Creates a new *APSAdminWebLoginHandler*.

Parameters

context- The bundle context.

public void setSessionIdFromRequestCookie(HttpServletRequest request)

Sets the session id from a cookie in the specified request.

Parameters

request- The request to get the session id cookie from.

public void setSessionIdFromRequestCookie(CookieTool.CookieReader cookieReader)

Sets the session id from a cookie in the specified request.

Parameters

cookieReader- The cookie reader to get the session id cookie from.

public void saveSessionIdOnResponse(HttpServletResponse response)

Saves the current session id on the specified response.

Parameters

response- The response to save the session id cookie on.

public void saveSessionIdOnResponse(CookieTool.CookieWriter cookieWriter)

Saves the current session id on the specified response.

Parameters

cookieWriter- The cookie writer to save the session id cookie on.

}

public *class* **APSLoginHandler** implements LoginHandler [se.natusoft.osgi.aps.tools.web] {

This class validates if there is a valid logged in user and also provides a simple login if no valid logged in user exists.

This utility makes use of APSAuthService to login auth and APSSessionService for session handling. Trackers for these services are created internally which requires the shutdown() method to be called when no longer used to cleanup.

The bundle needs to import the following packages for this class to work:

```
      se.natusoft.osgi.aps.api.auth.user;version="[0.9,2)",
      se.natusoft.osgi.aps.api.misc.session;version="[0.9,2)"
```

protected void setHandlerInfo(HandlerInfo handlerInfo)

Sets the handler info when not provided in constructor.

Parameters

handlerInfo- The handler info to set.

public void shutdown()

Since this class internally creates and starts service trackers this method needs to be called on shutdown to cleanup!

public String getLoggedInUser()

This returns the currently logged in user or null if none are logged in.

public boolean hasValidLogin()

Returns true if this handler sits on a valid login.

public boolean login(String userId, String pw)

Logs in with a userid and a password.

This method does not use or modify any internal state of this object! It only uses the APSAuthService that this object sits on. This allows code sitting on an instance of this class to use this method for validating a user without having to setup its own service tracker for the *APSAuthService* when this object is already available due to the code also being an *APSAdminWeb* member. It is basically a convenience.

Returns

true if successfully logged in, false otherwise.

Parameters

userId- The id of the user to login.

pw- The password of the user to login.

public boolean login(String userId, String pw, String requiredRole)

Logs in with a userid and a password, and a required role.

This method does not use or modify any internal state of this object! It only uses the APSAuthService that this object sits on. This allows code sitting on an instance of this class to use this method for validating a user without having to setup its own service tracker for the *APSAuthService* when this object is already available due to the code also being an *APSAdminWeb* member. It is basically a convenience.

Returns

a valid User object on success or null on failure.

Parameters

userId- The id of the user to login.

pw- The password of the user to login.

requiredRole- If non null the user is required to have this role for a successful login. If it doesn't null will

public *static interface* **HandlerInfo** [se.natusoft.osgi.aps.tools.web] {

Config values for the login handler.

String getSessionId()

Returns

An id to an APSSessionService session.

void setSessionId(String sessionId)

Sets a new session id.

Parameters

sessionId- The session id to set.

String getUserSessionName()

Returns

The name of the session data containing the logged in user if any.

String getRequiredRole()

Returns

The required role of the user for it to be considered logged in.

}

---

public *interface* **LoginHandler** [se.natusoft.osgi.aps.tools.web] {

This is a simple API for doing a login.

public boolean hasValidLogin()

Returns true if this handler sits on a valid login.

boolean login(String userId, String pw)

Logs in with a userid and a password.

Returns

true if successfully logged in, false otherwise.

Parameters

userId- The id of the user to login.

pw- The password of the user to login.

public void shutdown()

If the handler creates service trackers or other things that needs to be shutdown when no longer used this method needs to be called when the handler is no longer needed.

}

APSAuthService

This is a very simple little service that only does authentication of users. This service is currently used by the APS administration web (/apsadminweb) and APSExtProtocolHTTPTransportProvider for remote calls to services over http.

The idea behind this service is that it should be easy to provide an implementation of this that uses whatever authentication scheme you want/need. If you have an LDAP server you want to authenticate against for example, provide an implementation that looks up and authenticates the user against the LDAP server.

See this a little bit like an authentication plugin.

The APS web applications that use this only uses password authentication.

APSSimpleUserServiceAuthServiceProvider

This provides an APSAuthService that uses the APSSimpleUserService to authenticate users. It only supports password authentication. If you don't have your own implementation of APSAuthService then you can deploy this one along with APSSimpleUserService, and probably APSUserAdminWeb.

Please note however that the standard implementation of APSSimpleUserService can register several instances with an "instance=name" property where name is unique for each instance, and each instance can reference a different data source. This is configured under *persistence/dsrefs* in the configuration. If no instances are configured an instance of "aps-admin-web" will be created by default. If instances are configured the default will not be created. And now the the point: APSSimpleuserServiceAuthServiceProvider will as of now track the "aps-admin-web" instance of APSSimpleUserService! If no such instance is configured it will fail after a timeout of not finding a service!

API

public *interface* **APSAuthService<Credential>** [se.natusoft.osgi.aps.api.auth.user] {

This is intended to be used as a wrapper to other means of authentication. Things in APS that needs authentication uses this service.

Implementations can lookup the user in an LDAP for example, or use some other user service.

APS supplies an *APSSimpleUserServiceAuthServiceProvider* that uses the *APSSimpleUserService* to authenticate. It is provided in its own bundle.

Properties authUser(String userId, Credential credentials, AuthMethod authMethod) throws APSAuthMethodNotSupportedException

This authenticates a user. A Properties object is returned on successful authentication. null is returned on failure. The Properties object returned contains misc information about the user. It can contain anything or nothing at all. There can be no assumptions about its contents!

Returns

User properties on success, null on failure.

Parameters

userId- The id of the user to authenticate.

credentials- What this is depends on the value of AuthMethod. It is up to the service implementation to resolve this.

authMethod- This hints at how to interpret the credentials.

Throws

APSAuthMethodNotSupportedException- If the specified authMethod is not supported by the implementation.

Properties authUser(String userId, Credential credentials, AuthMethod authMethod, String role) throws APSAuthMethodNotSupportedException

This authenticates a user. A Properties object is returned on successful authentication. *null* is returned on failure. The Properties object returned contains misc information about the user. It can contain anything or nothing at all. There can be no assumptions about its contents!

Returns

User properties on success, null on failure.

Parameters

userId- The id of the user to authenticate.

credentials- What this is depends on the value of AuthMethod. It is up to the service implementation to resolve this.

authMethod- This hints at how to interpret the credentials.

role- The specified user must have this role for authentication to succeed. Please note that the APS admin webs will pass "apsadmin" for the role. The implementation might need to translate this to another role.

Throws

APSAuthMethodNotSupportedException- If the specified authMethod is not supported by the implementation.

AuthMethod[] getSupportedAuthMethods()

Returns an array of the AuthMethods supported by the implementation.

public *static enum* **AuthMethod** [se.natusoft.osgi.aps.api.auth.user] {

This hints at how to use the credentials.

NONE

Only userid is required.

PASSWORD

toString() on the credentials object should return a password.

KEY

The credential object is a key of some sort.

CERTIFICATE

The credential object is a certificate of some sort.

DIGEST

The credential object is a digest password.

SSO

The credential object contains information for participating in a single sign on.

```
}
```

APSSimpleUserService

This is an simple, easy to use service for handling logged in users. It provides two services: APSSimpleUserService and APSSimpleUserServiceAdmin. The latter handles all creation, editing, and deletion of roles and users. This service in itself does not require any authentication to use! Thereby you have to trust all code in the server! The APSUserAdminWeb WAB bundle however does require a user with role *apsadmin* to be logged in or it will simply repsond with a 401 (UNAUTHORIZED).

So why this and not org.osgi.service.useradmin ? Well, maybe I'm just stupid, but *useradmin* does not make sense to me. It seems to be missing things, specially for creating. You can create a role, but you cannot create a user. There is no obvious authentication of users. Maybee that should be done via the credentials Dictionary, but what are the expected keys in there ? APSSimpleUserService is intended to make user and role handling simple and clear.

Basic example

To login a user do something like this:

```
APSSimpleUserService userService ...
...
User user = userService.getUser(userId);
if (user == null) {
    throw new AuthException("Bad login!");
}
if (!userService.authenticateUser(user, password, APSSimpleUserService.AUTH_METHOD_PASSWORD)) {
    throw new AuthException("Bad login!");
}
...
if (user.isAuthenticated() && user.hasRole("apsadmin")) {
    ...
}
```

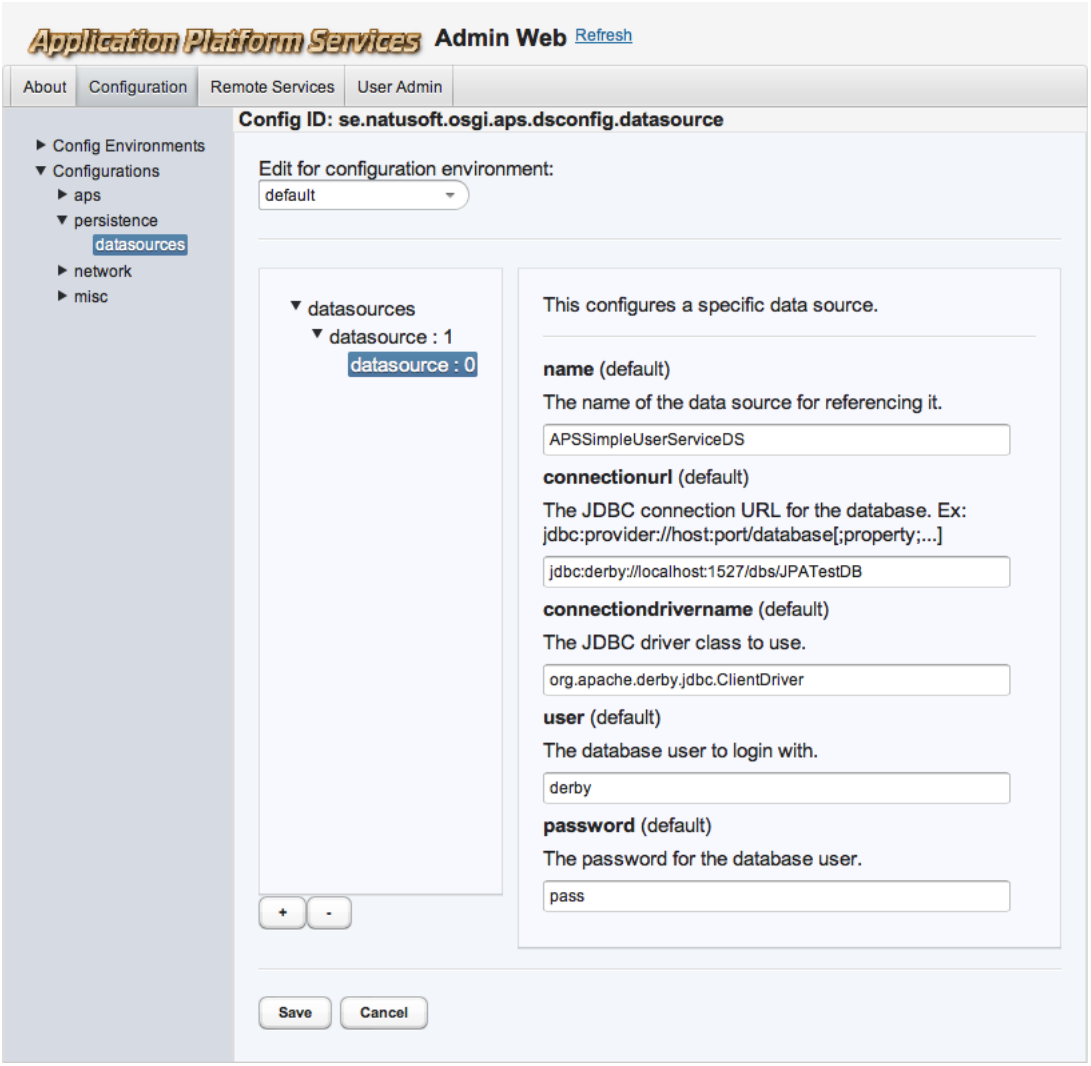
Setup

The following SQL is needed to create the database tables used by the service.

```
/*
 * This represents one role.
 */
create table role (
  /* The id and key of the role. */
  id varchar(50) not null primary key,

  /* A short description of what the role represents. */
  description varchar(200),

  /* 1 == master role, 0 == sub-role. */
  master int
);

/*
 * This represents one user.
 */
create table svcuser (
  /* User id and also key. */
  id varchar(50) not null primary key,

  /* For the provided implementation this is a password. */
  auth varchar(2000),

  /*
   * The service stores string properties for the user here as one long string.
   * These are not meant to be searchable only to provide information about the
   * user.
   *
   * You might want to adapt this size to the amount of data you will be adding
   * to a user.
   */
```

```
  user_data varchar(4000)
);

/*
 * A user can have one or more roles.
 */
create table user_role (
  user_id varchar(50) not null,
  role_id varchar(50) not null,
  primary key (user_id, role_id),
  foreign key (user_id) references svcuser (id),
  foreign key (role_id) references role (id)
);

/*
 * A role can have one ore more sub-roles.
 */
create table role_role (
  master_role_id varchar(50) not null,
  role_id varchar(50) not null,
  primary key (master_role_id, role_id),
  foreign key (master_role_id) references role (id),
  foreign key (role_id) references role (id)
);

/*
 * ---- This part is mostly an example ----
 * WARNING: You do however need a role called 'apsadmin' to be able to login to
 * /apsadminweb! The name of the user having that role does not matter. As long
 * as it is possible to login to /apsadminweb new roles and users can be created
 * there.
 */

/* The following adds an admin user. */
insert into role VALUES ('apsadmin', 'Default admin for APS', 1);
insert into svcuser VALUES ('apsadmin', 'admin', '');
insert into user_role VALUES ('apsadmin', 'apsadmin');

/* This adds a role for non admin users. */
insert into role VALUES ('user', 'Plain user', 1);
```

After the tables have been created you need to configure a datasource for it in /apsadminweb configuration tab:



Please note that the above picture is just an example. The data source name *APSSimpleUserServiceDS* in this example should be configured in the *persistence/dsRefs* config where you provide a name and a datasource reference. The service will be looking up the entry with that name, and use the specified datasource! For example:

```
name: aps-admin-web
dsRef: APSSimpleUserServiceDS
```

This example happens to be the default if no instances have been configured and is required if you want to use authentication for the APS admin web. You should probably define your own instance if you are going to use this service. The *dsRef* part is exactly the same name as defined in the data source configuration (*persistence/datasources*).

The rest of the datasource entry in the picture above depends on your database and where it is running. Also note that the "(default)" after the field names in the above picture are the name of the currently selected configuration environment. This configuration is configuration environment specific. You can point out different database servers for different environments for example.

When the datasource is configured and saved then you can go to *"Configuration tab, Configurations/aps/adminweb"* and enable the "requireauthentication" config. **If you do this before setting up the datasource and you have choosen to use the provided implementation of APSAuthService that uses APSSimpleUserService to login then you will be completely locked out!**

Troubleshooting

If you have managed to lock yourself out of /apsadminweb as described above then I suggest editing the *APSFilesystemService root* /filesystems/se.natusoft.osgi.aps.core.config.service.APSConfigServiceProvider/apsconfig-se.natusoft.aps.adminweb-1 file and changeing the following line:

```
se.natusoft.aps.adminweb_1.0_requireauthentication=true
```

to *false* instead. Then restart the server. Also se the APSFilesystemService documentation for more information. The APSConfigService is using that service to store its configurations.

JDBC Drivers

There is a catch with OSGi and its classpath isolation. The *APSSimpleUserService* makes use of the *APSJPAService* whose implementation *APSOpenJPAProvider* cheats OSGi a bit by using *MultiBundleClassLoader* (is available in aps-tools-library bundle) and merges the service classpath with the client classpath which is a requirement for the JPA framework to work (it needs access to both framework code in the service classpath and client entities in the client classpath). This also has the side effect that the client can provide a JDBC driver in its bundle. The *APSSimpleUserService* do provide a JDBC driver for *Derby 10.9.1.0*.

Another catch with this is that users of *APSSimpleUserService* are not part of this collective classpath and can thereby not make drivers available in their bundles, or at least not right off, there is however a workaround to this. There is a natsty way that you can pass on the client bundle class loader right through the *APSSimpleUserService* to *APSJPAService* by creating an instance of *MultiBundleClassLoader* and set it as context classloader:

```
MultiBundleClassLoader mbClassLoader = new
    MultiBundleClassLoader(bundleContext.getBundle());
Thread.currentThread().setContextClassLoader(mbClassLoader);
```

Do this before the first call to *APSSimpleUserService*. The *APSJPAService* will check if the current context class loader is a MultiBundleClassLoader and if so extract the bundles from it and add to its own MultiBundleClassLoader. This way you have extended the classpath that the JPA framework will se to 3 bundles: aps-openjpa-provider, aps-simple-user-service-provider, and your client bundle, which can then contain a JDBC driver.

The catches are unfortunately not over yet! You also need to configure your own instance of *APSSimpleUserService* with its own data source in the configuration, and your client needs to add the name of this configuration to the tracker for the *APSSimpleUserService* :

```
APSServiceTracker<APSSimpleUserService> userServiceTracker =
    new APSServiceTracker<>(bundleContext, APSSimpleUserService.class, "(instance=instName)", "30
    seconds");
```

or

```
@OSGiService(additionalSearchCriteria="(instance=instName)", timeout="30 seconds")
APSSimpleUserService userService;
```

where *instName* is whatever name you gave the instance in the configuration. Then try to have only one bundle call this service since each different bundle calling the service will extend the service classpath with that bundle!

APIs

public *interface* **APSSimpleUserService** [se.natusoft.osgi.aps.api.auth.user] {

This is the API of a simple user service that provide basic user handling that will probably be enough

in many cases, but not all.

Please note that this API does not declare any exceptions! In the case of an exception being needed the APSSimpleUserServiceException should be thrown. This is a runtime exception.

public static final String AUTH_METHOD_PASSWORD = "password"

Password authentication method for authenticateUser().

public Role getRole(String roleId)

Gets a role by its id.

Returns

A Role object representing the role or null if role was not found.

Parameters

roleId- The id of the role to get.

public User getUser(String userId)

Gets a user by its id.

Returns

A User object representing the user or null if userId was not found.

Parameters

userId- The id of the user to get.

public boolean authenticateUser(User user, Object authentication, String authMethod)

Authenticates a user using its user id and user provided authentication.

Returns

true if authenticated, false otherwise. If true user.isAuthenticated() will also return true.

Parameters

user- The User object representing the user to authenticate.

authentication- The user provided authentication data. For example if AuthMethod is AUTH_METHOD_PASSWORD

authMethod- Specifies what authentication method is wanted.

}

---

public *interface* **APSSimpleUserServiceAdmin** extends APSSimpleUserService [se.natusoft.osgi.aps.api.auth.user] {

Admin API for APSSimpleUserService.

public RoleAdmin createRole(String name, String description)

Creates a new role.

Returns

a new Role object representing the role.

Parameters

name- The name of the role. This is also the key and cannot be changed.

description- A description of the role. This can be updated afterwards.

public void updateRole(Role role)

Updates a role.

Parameters

role- The role to update.

public void deleteRole(Role role)

Deletes a role.

Parameters

role- The role to delete. This will likely fail if there are users still having this role!

public List<RoleAdmin> getRoles()

Returns all available roles.

public UserAdmin createUser(String id)

Creates a new user. Please note that you get an empty user back. You probably want to add roles and also possibly properties to the user. After you have done that call *updateUser(user)*.

Returns

A User object representing the new user.

Parameters

id- The id of the user. This is key so it must be unique.

public void updateUser(User user)

Updates a user.

Parameters

user- The user to update.

public void deleteUser(User user)

Deletes a user.

Parameters

user- The user to delete.

public List<UserAdmin> getUsers()

Returns all users.

public void setUserAuthentication(User user, String authentication)

Sets authentication for the user.

Parameters

user- The user to set authentication for.

authentication- The authentication to set.

}

---

public *class* **APSAuthMethodNotSupportedException** extends APSRuntimeException [se.natusoft.osgi.aps.api.auth.user.exceptions] {

This is thrown by APSAuthService when the implementation does not support the selected auth method.

public APSAuthMethodNotSupportedException(String message)

Creates a new APSAuthMethodNotSupportedException instance.

Parameters

message- The exception messaging.

public APSAuthMethodNotSupportedException(String message, Throwable cause)

Creates a new APSAuthMethodNotSupportedException instance.

Parameters

message- The exception messaging.

cause- The exception that is the cause of this one.

}

---

public *class* **APSSimpleUserServiceException** extends APSRuntimeException [se.natusoft.osgi.aps.api.auth.user.exceptions] {

Indicates a problem with the APSSimpleUserService.

public APSSimpleUserServiceException(String message)

Creates a new APSSimpleUserServiceException instance.

Parameters

message- The exception messaging.

public APSSimpleUserServiceException(String message, Throwable cause)

Creates a new APSSimpleUserServiceException instance.

Parameters

message- The exception messaging.

cause- The cause of the exception.

}

---

public *interface* **Role** extends Comparable<Role> [se.natusoft.osgi.aps.api.auth.user.model] {

This defines a role.

public String getId()

Returns

The id of the role.

public String getDescription()

Returns

A description of the role.

public boolean hasRole(String roleName)

Returns true if the role has the specified sub role name.

Parameters

roleName- The name of the role to check for.

boolean isMasterRole()

Returns

true if this role is a master role. Only master roles can be added to users.

}

---

public *interface* **RoleAdmin** extends Role [se.natusoft.osgi.aps.api.auth.user.model] {

Provides update API for Role.

public void setDescription(String description)

Changes the description of the role.

Parameters

description- The new description.

public List<Role> getRoles()

Returns all sub roles for this role.

public void addRole(Role role)

Adds a sub role to this role.

Parameters

role- The role to add.

public void removeRole(Role role)

Removes a sub role from this role.

Parameters

role- The role to remove.

public void setMasterRole(boolean masterRole)

Sets whether this is a master role or not.

Parameters

masterRole- true for master role.

}

---

public *interface* **User** extends Comparable<User> [se.natusoft.osgi.aps.api.auth.user.model] {

This defines a User.

public String getId()

Returns the unique id of the user.

public boolean isAuthenticated()

Returns true if this user is authenticated.

public boolean hasRole(String roleName)

Returns true if the user has the specified role name.

Parameters

roleName- The name of the role to check for.

public Properties getUserProperties()

This provides whatever extra information about the user you want. How to use this is upp to the user of the service. There are some constants in this class that provide potential keys for the user properties.

Please note that the returned properties are read only!

public static final String USER_NAME = "name"

Optional suggestion for user properties key.

public static final String USER_PHONE = "phone"

Optional suggestion for user properties key.

public static final String USER_PHONE_WORK = "phone.work"

Optional suggestion for user properties key.

public static final String USER_PHONE_HOME = "phone.home"

Optional suggestion for user properties key.

public static final String USER_EMAIL = "email"

Optional suggestion for user properties key.

}

---

public *interface* **UserAdmin** extends User [se.natusoft.osgi.aps.api.auth.user.model] {

Provides update API for the User.

public List<Role> getRoles()

Returns all roles for this user.

public void addRole(Role role)

Adds a role to this user.

Parameters

role- The role to add.

public void removeRole(Role role)

Removes a role from this user.

Parameters

role- The role to remove.

public void addUserProperty(String key, String value)

Adds a user property.

Parameters

key- The key of the property.

value- The value of the property.

public void removeUserProperty(String key)

Removes a user property.

Parameters

key- The key of the property to remove.

public void setUserProperties(Properties properties)

Sets properties for the user.

To update the user properties either first do *getProperties()*, do your changes, and then call this method with the changed properties or just use the *addUserProperty()* and *removeUserProperty()* methods.

Parameters

properties- The properties to set.

```
}
```

APSDataSource

This is a service that provides named data source definitions. It does **not** provide pooled *javax.sql.DataSource* instances! It only provides definitions with connection url, driver name, user and password. This service can be used by other services that provide DataSource pooling for example. The APSSimpleUserServiceProvider makes use of this service by looking up ”APSSimpleUserServiceDS” passing the information on to the APSJPAService in its properties. Not everything can make use of an *javax.sql.DataSource*, but everything can make use of the information provided by this service.

The actual data source definitions are configured in the */apsadminweb* under configuration group ”persistence”.

APIs

The complete APS javadoc can be found at http://apidoc.natusoft.se/APS/.

public *interface* **DataSourceDef** [se.natusoft.osgi.aps.api.data.jdbc.model] {

This represents information required for setting upp a JDBC data source.

String getName()

Returns

The name of this data source definition. This information is optional and can return null!

String getConnectionURL()

Returns

The JDBC connection URL. Ex: jdbc:provider://host:port/database[;properties].

String getConnectionDriveName()

Returns

The fully qualified class name of the JDBC driver to use.

String getConnectionUserName()

Returns

The name of the database user to login as.

String getConnectionPassword()

Returns

The password for the database user.

}

---

public *interface* **APSDataSourceDefService** [se.natusoft.osgi.aps.api.data.jdbc.service] {

This service provides lookup of configured data source definitions. These can be used to setup connection pools, JPA, ...

DataSourceDef lookupByName(String name)

Looks up a data source definition by its configured name.

Returns

A DataSourceDef or null if name was not valid.

Parameters

name- The name to lookup.

List<DataSourceDef> getAllDefinitions()

Returns

All available definitions.

}

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 restared 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 *APSDataSourceDefService* 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.

public *static interface* **APSJPAEntityManagerProvider** [se.natusoft.osgi.aps.api.data.jpa.service] {

Once you get this it is valid until the *APSJPAService* is stopped (which will happen if the service is redeployed!).

public 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!

}

APSJSONService

This provides exactly the same functionallity as APSJSONLib. It actually wraps the library as a service. The reason for that is that I wanted to be able to redeploy the library without forcing a redeploy of the Bunde using it. A redeploy of the library will force a redeploy of this service, but not the client of this service. The APS clients of this service uses APSServiceTracker wrapped as a service and thus handles this service leaving and returning without having to care about it.

This service and the library existrs for internal use. It is here and can be used by anyone, but in most cases like serializing java beans back and forth to JSON (which this can do) Jacksson would still be a better choice and offers more flexibility. In the long run I’m going to see if I can replace the internal use of this with Jacksson as well.

aps-persistent-named-queue-provider

This provides an implementation of **APSNamedQueueService** that makes use of **APSFilesystemService** for storage. Clients just create or get a queue by a unique name. It is then possible to push bytes to the queue or to pull bytes from the queue. It is rather simple.

APIS

public *interface* **APSNamedQueueService** [se.natusoft.osgi.aps.api.misc.queue] {

A named queue as a service. How long lived it is depends on the implementation.

Note that there can be only one receiver per queue. Once an item is delivered it is gone from the queue!

APSQueue createQueue(String name) throws APSIOException

Creates a new queue.

Parameters

name- The name of the queue to create. If the named queue already exists, it is just returned,

Throws

APSIOException- on failure to create queue.

void removeQueue(String name) throws APSResourceNotFoundException

Removes the named queue.

Parameters

name- The name of the queue to remove.

Throws

APSResourceNotFoundException- on failure to remove the queue.

APSQueue getQueue(String name) throws APSResourceNotFoundException

Returns the named queue. If it does not exist, it is created.

Parameters

name- The name of the queue to get.

Throws

APSResourceNotFoundException- on failure to get queue.

}

---

void push(byte[] item) throws APSIOException

Pushes a new item to the end of the list.

Parameters

item- The item to add to the list.

Throws

APSIOException- on any failure to do this operation.

byte[] pull(long timeout) throws APSIOTimeoutException

Pulls the first item in the queue, removing it from the queue.

Returns

The pulled item.

Parameters

timeout- A value of 0 will cause an immediate APSIOException if the queue is empty. Any

Throws

APSIOException- on any failure to do this operation.

byte[] peek() throws APSIOException, UnsupportedOperationException

Looks at, but does not remove the first item in the queue.

Returns

The first item in the queue.

Throws

APSIOException- on any failure to do this operation.

UnsupportedOperationException- If this operation is not supported by the implementation.

int size() throws APSIOException, UnsupportedOperationException

Returns the number of items in the queue.

Throws

APSIOException- on any failure to do this operation.

UnsupportedOperationException- If this operation is not supported by the implementation.

boolean isEmpty() throws APSIOException

Returns true if this queue is empty.

Throws

APSIOException- on any failure to do this operation.

void release()

Releases this APSQueue instance to free up resources. After this call this specific instance will be invalid and a new one have to be gotten from APSNamedQueueService.

}

APSResolvingBundleDeployer

This is a bundle deployer that is intended as an alternative to the server provided deployer.

This bundle deployer will try to automatically resolve deploy dependencies. It does this by having a fail threshold. If the deploy of a bundle fails it just keeps quiet and put the bundle at the end of the list of bundles to deploy. It updates the try count for the bundle however. Next time the bundle is up for deploy it might have the dependencies it needs and will deploy. If not it goes back to the end of the list again and its retry count is incremented again. This repeats until the retry count reaches the threshold value in which case an error is logged and the bundle will not be attempted to be deployed again unless it gets a new timestamp on disk.

Glassfish does something similar, but Virgo fails completely unless bundles are deployed in the correct order. You have to provide a par file for Virgo to deploy correctly.

There is one catch to using this deployer: It does not handle WAB bundles! Neither Glassfish nor Virgo seems to handle WAB deployment using the OSGi extender pattern. If they did they would recognize a WAB being deployed even though it is deployed by this deployer and handle it. They dont!

Configuration

The following configuration is available for this deployer. Edit this in /apsadminweb ”Configurations” tab under the *aps* node.

deployDirectory - The directory to deploy bundles from. All bundles in this directory will be attempted to be deployed.

failThreshold - The number of failed deploys before giving upp. The more bundles and the more dependencies among them the higher the value should be. The default value is 8.

APSSessionService

This service provides session storage functionality. You can create a session, get an existing session by its id, and close a session. Each session can hold any number of named objects.

Why a session service ? To begin with, this is not an HttpSession! That said, it was created to handle a single session among several web applications. This for the APS administration web which are made up of several web applications working toghether. This is explained in detail in the APSAdminWeb documentation.

APIs

public *interface* **APSSession** [se.natusoft.osgi.aps.api.misc.session] {

This represents an active session.

String getId()

Returns

The id of this session.

boolean isValid()

Returns

true if this session is still valid.

void saveObject(String name, Object object)

Saves an object in the session. Will do nothing if the session is no longer valid.

Parameters

name- The name to store the object under.

object- An object to store in the session.

Object retrieveObject(String name)

Returns a object stored under the specified name or null if no object is stored under that name.

If isValid() returns false then this will always return null.

Parameters

name- The name of the object to get.

}

---

public *interface* **APSSessionService** [se.natusoft.osgi.aps.api.misc.session] {

This is not a http session! It is a simple session that can be used by any code running in the same OSGi server.

APSSession createSession(int timeoutInMinutes)

Creates a new session.

Parameters

timeoutInMinutes- The timeout in minutes.

APSSession createSession(String sessionId, int timeoutInMinutes)

Creates a new session.

The idea behind this variant is to support distributed sessions. The implementation must use a session id that is unique enough to support this. The APS implementation uses java.util.UUID.

Parameters

sessionId- The id of the session to create.

timeoutInMinutes- The timeout in minutes.

APSSession getSession(String sessionId)

Looks up an existing session by its id.

Returns

A valid session having the specified id or null.

Parameters

sessionId- The id of the session to lookup.

void closeSession(String sessionId)

Closes the session represented by the specified id. After this call APSSession.isValid() on an *APSSession* representing this session will return false.

Parameters

sessionId- The id of the session to leaveSyncGroup.

}

APSDefaultDiscoveryServiceProvider

This is a simple service where you can publish and unpublish service information. This information will be distributed to all other nodes that are configured to be part of the disovery.

Service data can be configured to be shared by Multicast, or TCP. The *APSTCPIPService* is used to send and receive data.

Discovery information

The discovery information is just a Properties object. Anything you want can be put into it, but the following keys are suggested for interoperability:

```
public class DiscoveryKeys {

    /** A name of the service. */
    public static final String NAME = "name";

    /** The version of the service. */
    public static final String VERSION = "version";

    /** An URI as used by APSTcpipService. */
    public static final String APS_URI = "apsURI";

    /** A URL for accessing the service. */
    public static final String URL = "url";

    /** The port of the service. */
    public static final String PORT = "port";

    /** The host of the service. */
    public static final String HOST = "host";

    /** An informative description of the service. */
    public static final String DESCRIPTION = "description";

    /** This is used by APSClusterService to announce cluster members. */
    public static final String APS_CLUSTER_NAME = "apsClusterName";

    /** The protocol of the service, like TCP, UDP, Multicast */
    public static final String PROTOCOL = "protocol";

    /** Some description of the type of the content provided by the service. */
    public static final String CONTENT_TYPE = "contentType";

    /** A timestamp of when the entry was last updated. */
    public static final String LAST_UPDATED = "lastUpdated";
}
```

Transport data format

The data is transported over the network in JSON format:

```
{
    action: "ADD" / "REMOVE",
    serviceDescription: {
        name: "myservice",
        version: "1.5.4",
        apsURI: "tcp://myhost:5564"
        ...
    }
}
```

API

```
/**
 * A network service discovery.
 *
 * There a many such services available in general, a bit less from a java
 * perspective, but the intention with this is not to compete with any of
 * the others, but to provide an extremely simple way to discover remote
```

```
     * services in an as simple an primitive way as possible. Basically a way
     * to have multiple hosts running APS based code find each other easily,
     * may it be by simple configuration or by multicast or TCP, or wrapping
     * some other service.
     */
    public interface APSSimpleDiscoveryService {

        //
        // Methods
        //

        /**
         * On a null filter all services are returned. The filter is otherwise
         * of LDAP type: (&(this=that)(something=pizza)).
         *
         * @param filter The filter to narrow the results.
         */
        Set<Properties> getServices(String filter);

        /**
         * Publishes a local service. This will announce it to other known
         * APSSimpleDiscoveryService instances.
         *
         * @param serviceProps This is a set of properties describing the
         *                     service. There are some suggested keys in
         *                     DiscoveryKeys for general compatibility.
         *
         * @throws APSDiscoveryException on problems to publish (note:
         *                               this is a runtime exception!).
         */
        void publishService(Properties serviceProps) throws APSDiscoveryException;

        /**
         * Recalls the locally published service, announcing to other known
         * APSSimpleDiscoveryService instances that this service is no longer available.
         *
         * @param unpublishFilter An LDAP type filter that matches an entry or entries
         *                        to unpublish. Any non locally published services cauth
         *                        in the filter will be ignored.
         *
         * @throws APSDiscoveryException on problems to publish (note: this is a
         *                               runtime exception!).
         */
        void unpublishService(String unpublishFilter) throws APSDiscoveryException;
    }
```

APSExternalProtocolExtender

This is an OSGi bundle that makes use of the OSGi extender pattern. It listens to services being registered and unregistered and if the services bundles *MANIFEST.MF* contains `APS-Externalizable: true` all services published by the bundle is made externally available. If the *MANIFEST.MF* contains `APS-Externalizable: false` however making services externally available is forbidden. It is also possible as an alternative to true/false specify a list of fully qualified service names to make only those services externally available. This overrides any other specification.

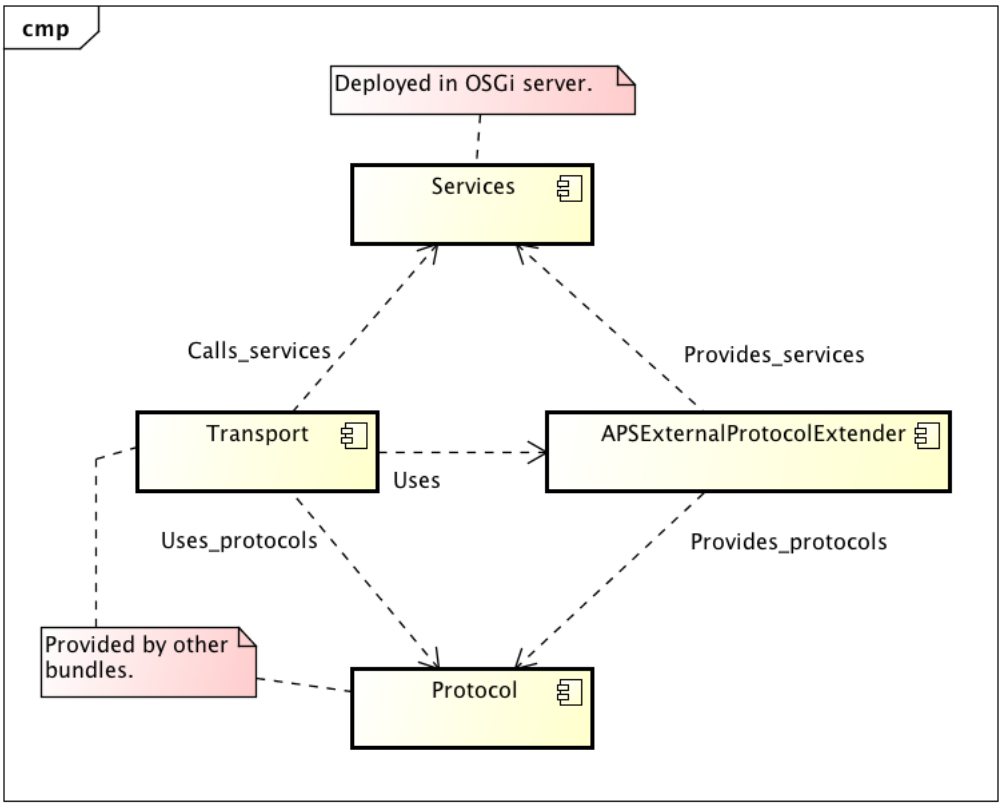
A specific service can also be registered containing an *aps-externalizable* property with value *true* to be externalizable. If your bundle uses APSActivator (APSToolsLib) as bundle activator then any of @APSExternalizable and @APSRemoteService on the class will make APSActivator set the *aps-externalizable* property to true when registering the service.

The exernal protocol extender also provides a configuration where services can be specified with their fully qualified name to be made externally available. If a bundle however have specifically specified false for the above manifest entry then the config entry will be ignored.

So, what is meant by *made externally available* ? Well what this bundle does is to analyze with reflection all services that are in one way or the other specified as being externalizable (manifest or config) and for all callable methods of the service an *APSExternallyCallable* object will be created and saved locally with the service name. *APSExternallyCallable* extends *java.util.concurrent.Callable*, and adds the possibility to add parameters to calls and also provides meta data for the service method, and the bundle it belongs to. There is also an *APSRESTCallable* that extends *APSExternallyCallable* and also takes an http method and maps that to a appropriate service method.

The overall structure

The complete picture for making services externally callable looks like this:



This bundle provides the glue between the services and the protocols. Transports and protocols have to be provided by other bundles.

The flow is like this:

1. Transport gets some request and an InputStream.
2. Transport gets some user selected protocol (The APSExtProtocolHTTPTransportProvider allows specification of both protocol, protocol version, and service to call in the URL).
3. Transport calls *APSExternalProtocolService* to get requested protocol.
4. Transport calls protocol to parse InputStream and it returns an *RPCRequest*.
5. Transport uses the information in the RPCRequest to call a service using *APSExternalProtocolService*.
6. Transport takes the result from the call and passes to the protocol along with an OutputStream to write response on.

APSExternalProtocolService

This bundle registers an *APSExternalProtocolService* that will provide all *APSExternallyCallable* instances (or rather copies of them since you can modify the one you get back by providing arguments). This service also provides getters for available remote protocols and you can register with it to receive information about chages for services and protocols.

Protocols

There is a base API for protocols: *RPCProtocol*. APIs for different types of protocols should extend this. The protocol type APIs are service APIs and services implementing them must be provided by other bundles. This bundle looks for and keeps track of all such service providers.

The *StreamedRPCProtocol* extends *RPCProtocol* and provides a method for parsing a request from an *InputStream* returning an *RPCRequest* object. This request object contains the name of the service, the method, and the parameters. This is enough for using *APSExternalProtocolService* to do a call to the service. The request object is also used to write the call response on an OutputStream. There is also a method to write an error response.

It is the responsibility of the transport provider to use a protocol to read and write requests and responses and to use the request information to call a service method. An exception is the case of http transports supporting REST that must take the responibility for returning an http status.

Getting information about services and protocols.

A transport provider can register themselves with the *APSExternalProtocolService* by implementing the *APSExternalProtocolListener* interface. They will then be notified when a new externalizable service becomes available or is leaving and when a protocol becomes available or is leaving.

WARNING - Non backwards compatible changes!

This version have non backwards compatible changes! *StreamedRPCProtocol* have changed in parameters for *parseRequest(...)* and *isRest()* is gone. *RPCProtocol* have changes in parameters for crateRPCError(...). The error code is now gone. These changes was a necessity! The old was really bad and tried to solve REST support in a very stupid way. It is now handled very much more elegantly without any special support for it with *is*methods!

The *APSExtProtocolHTTPTransportProvider* now checks if an *RPCError* (returned by createRPCError(...)) object actually is an *HTTPError* subclass providing an HTTP error code to return.

parseRequest(...) parameters now also contain the class of the service and a new RequestIntention enum. The service class is only for inspecting methods for annoations or other possible meta data. The JSONREST protocol for example uses this to find annotations indicating GET, PUT, DELETE, etc methods, which is far more flexible than the old solution of requiring a get(), put(), etc method. The RequestIntention enum provides the following values: CREATE, READ, UPDATE, DELETE, UNKNOWN. That is CRUD + UNKNOWN. It will be UNKNOWN if the transport cannot determine such information. These are basically to support REST protocols without being too HTTP specific. Other transports can possible also make use of them.

See also

APSExtProtocolHTTPTransportProvider - Provides a HTTP transport.

APSStreamedJSONRPCProtocolProvider - Provides version 1.0 and 2.0 of JSONRPC, JSONHTTP and JSONREST.

APIs

public *interface* **APSExternalProtocolService** [se.natusoft.osgi.aps.api.external.extprotocolsvc] {

This service makes the currently available externalizable services available for calling. It should be used by a bundle providing an externally available way of calling a service (JSON over http for example) to translate and forward calls to the local service. The locally called service is not required to be aware that it is called externally.

Never cache any result of this service! Always make a new call to get the current state. Also note that it is possible that the service represented by an APSExternallyCallable have gone away after it was returned, but before you do call() on it! In that case an APSNoServiceAvailableException will be thrown. Note that you can register as an APSExternalProtocolListener to receive notifications about externalizable services coming and going, and also protocols coming and going to keep up to date with the current state of things.

public Set<String> getAvailableServices()

Returns all currently available services.

public List<APSExternallyCallable> getCallables(String serviceName)

Returns all APSExternallyCallable for the named service object.

Parameters

serviceName- The name of the service to get callables for.

public Set<String> getAvailableServiceFunctionNames(String serviceName)

Returns the names of all available functions of the specified service.

Parameters

serviceName- The service to get functions for.

public APSExternallyCallable getCallable(String serviceName, String serviceFunctionName)

Gets an APSExternallyCallable for a specified service name and service function name.

Returns

An APSExternallyCallable instance or null if the combination of service and serviceFunction is not available.

Parameters

serviceName- The name of the service object to get callable for.

serviceFunctionName- The name of the service function of the service object to get callable for.

public List<RPCProtocol> getAllProtocols()

Returns

All currently deployed providers of RPCProtocol.

public RPCProtocol getProtocolByNameAndVersion(String name, String version)

Returns an RPCProtocol provider by protocol name and version.

Returns

Any matching protocol or null if nothing matches.

Parameters

name- The name of the protocol to get.

version- The version of the protocol to get.

public List<StreamedRPCProtocol> getAllStreamedProtocols()

Returns

All currently deployed providers of StreamedRPCProtocol.

public StreamedRPCProtocol getStreamedProtocolByNameAndVersion(String name, String version)

Returns a StreamedRPCProtocol provider by protocol name and version.

Returns

Any matching protocol or null if nothing matches.

Parameters

name- The name of the streamed protocol to get.

version- The version of the streamed protocol to get.

public void addExternalProtocolListener(APSExternalProtocolListener externalServiceListener)

Add a listener for externally available services.

Parameters

externalServiceListener- The listener to add.

public void removeExternalProtocolListener(APSExternalProtocolListener externalServiceListener)

Removes a listener for externally available services.

Parameters

externalServiceListener- The listener to remove.

}

---

public *interface* **APSExternallyCallable<ReturnType>** extends Callable<ReturnType> [se.natusoft.osgi.aps.api.external.extprotocolsvc.model] {

This API represents one callable service method.

public String getServiceName()

Returns

The name of the service this callable is part of.

public String getServiceFunctionName()

Returns

The name of the service function this callable represents.

public DataTypeDescription getReturnDataDescription()

Returns

A description of the return type.

public List<ParameterDataTypeDescription> getParameterDataDescriptions()

Returns

A description of each parameter type.

public Bundle getServiceBundle()

Returns

The bundle the service belongs to.

public Class getServiceClass()

Returns the class of the service implementation.

ReturnType call(Object... arguments) throws Exception

Calls the service method represented by this APSExternallyCallable.

Returns

The return value of the method call if any or null otherwise.

Parameters

arguments- Possible arguments to the call.

Throws

Exception- Any exception the called service method threw.

}

---

public *interface* **APSExternalProtocolListener**
[se.natusoft.osgi.aps.api.external.extprotocolsvc.model] {

A listener for externally available services. Please note that this means that the service is available for potential external protocol exposure! For it to be truly available there also has to be a protocol and transport available. It is probably only transports that are interested in this information!

public void externalServiceAvailable(String service, String version)

This gets called when a new externally available service becomes available.

Parameters

service- The fully qualified name of the newly available service.

version- The version of the service.

public void externalServiceLeaving(String service, String version)

This gets called when an externally available service no longer is available.

Parameters

service- The fully qualified name of the service leaving.

version- The version of the service.

public void protocolAvailable(String protocolName, String protocolVersion)

This gets called when a new protocol becomes available.

Parameters

protocolName- The name of the protocol.

protocolVersion- The version of the protocol.

public void protocolLeaving(String protocolName, String protocolVersion)

This gets called when a new protocol is leaving.

Parameters

protocolName- The name of the protocol.

protocolVersion- The version of the protocol.

}

---

public *interface* **APSRESTCallable** extends APSExternallyCallable [se.natusoft.osgi.aps.api.external.extprotocolsvc.model] {

This is a special variant of APSExternallyCallable that supports a HTTP REST call.

This is only available when a service have zero or one method whose name starts with put, zero or one method whose name starts with post, and so on. There has to be at least one method of put, post, get or delete.

APSExternalProtocolService can provide an instance of this is a service matches the criteria.

This is only of use for HTTP transports! aps-ext-protocol-http-transport-provider does make use of this for protocols that indicate they support REST.

public boolean supportsPut()

Returns

true if the service supports the PUT method.

public boolean supportsPost()

Returns

true if the service supports the POST method.

public boolean supportsGet()

Returns

true if the service supports the GET method.

public boolean supportsDelete()

Returns

true if the service supports the DELETE method.

public void selectMethod(HttpMethod method)

This selects the method to call with this callable.

Parameters

method- The selected method to call.

public *static enum* **HttpMethod** [se.natusoft.osgi.aps.api.external.extprotocolsvc.model] {

This defines the valid choices for selectMethod(...).

}

}

---

public *class* **APSRESTException** extends APSRuntimeException
[se.natusoft.osgi.aps.api.net.rpc.errors] {

This is a special exception that services can throw if they are intended to be available as REST services through the aps-external-protocol-extender + aps-ext-protocol-http-transport-provider. This allows for better control over status codes returned by the service call.

public APSRESTException(int httpStatusCode)

Creates a new *APSRESTException*.

Parameters

httpStatusCode- The http status code to return.

public APSRESTException(int httpStatusCode, String message)

Creates a new *APSRESTException*.

Parameters

httpStatusCode- The http status code to return.

message- An error messaging.

public int getHttpStatusCode()

Returns the http status code.

}

---

public *enum* **ErrorType** [se.natusoft.osgi.aps.api.net.rpc.errors] {

This defines what I think is a rather well though through set of error types applicable for an RPC call. No they are not mine, they come from Matt Morley in his JSONRPC 2.0 specification at http://jsonrpc.org/spec.html.

I did however add the following:

• SERVICE_NOT_FOUND - Simply because this can happen in this case!
• AUTHORIZATION_REQUIRED - This is also a clear possibility.
• BAD_AUTHORIZATION

PARSE_ERROR

Invalid input was received by the server. An error occurred on the server while parsing request data.

INVALID_REQUEST

The request data sent is not a valid.

METHOD_NOT_FOUND

The called method does not exist / is not available.

SERVICE_NOT_FOUND

The called service does not exist / is not available.

INVALID_PARAMS

The parameters to the method are invalid.

INTERNAL_ERROR

Internal protocol error.

SERVER_ERROR

Server related error.

AUTHORIZATION_REQUIRED

Authorization is required, but none was supplied.

BAD_AUTHORIZATION

Bad authorization was supplied.

}

---

public *interface* **HTTPError** extends RPCError [se.natusoft.osgi.aps.api.net.rpc.errors] {

Extends *RPCError* with an HTTP status code. HTTP transports can make use of this information.

public int getHttpStatusCode()

Returns

Returns an http status code.

}

---

public *interface* **RPCError** [se.natusoft.osgi.aps.api.net.rpc.errors] {

This represents an error in servicing an RPC request.

public ErrorType getErrorType()

The type of the error.

public String getErrorCode()

A potential error code.

public String getMessage()

Returns an error messaging. This is also optional.

public boolean hasOptionalData()

True if there is optional data available. An example of optional data would be a stack trace for example.

public String getOptionalData()

The optional data.

}

---

public *class* **RequestedParamNotAvailableException** extends APSRuntimeException [se.natusoft.osgi.aps.api.net.rpc.exceptions] {

This exception is thrown when a parameter request cannot be fulfilled.

public RequestedParamNotAvailableException(String message)

Creates a new *RequestedParamNotAvailableException* instance.

Parameters

message- The exception messaging.

public RequestedParamNotAvailableException(String message, Throwable cause)

Creates a new *RequestedParamNotAvailableException* instance.

Parameters

message- The exception messaging.

cause- The cause of this exception.

}

---

public *abstract class* **AbstractRPCRequest** implements RPCRequest [se.natusoft.osgi.aps.api.net.rpc.model] {

This provides a partial implementation of RPCRequest.

public AbstractRPCRequest(String method)

Creates a new AbstractRPCRequest.

Parameters

method- The method to call.

public AbstractRPCRequest(RPCError error)

Creates a new AbstractRPCRequest.

Parameters

error- An RPCError indicating a request problem, most probably of ErrorType.PARSE_ERROR type.

public AbstractRPCRequest(String method, Object callId)

Creates a new AbstractRPCRequest.

Parameters

method- The method to call.

callId- The callId of the call.

protected Map<String, Object> getNamedParameters()

Returns

The named parameters.

protected List<Object> getParameters()

Returns

The sequential parameters.

public void setServiceQName(String serviceQName)

Sets the fully qualified name of the service to call. This is optional since not all protocol delivers a service name this way.

Parameters

serviceQName- The service name to set.

public void addParameter(Object parameter)

Adds a parameter. This is mutually exclusive with addParameter(name, parameter)!

Parameters

parameter- The parameter to add.

}

---

public *enum* **RequestIntention** [se.natusoft.osgi.aps.api.net.rpc.model] {

The intention of a request.

}

---

public *interface* **RPCExceptionConverter** [se.natusoft.osgi.aps.api.net.rpc.model] {

An instance of this can be passed to RPCRequest to convert the cauth exception to an RPCError.

RPCError convertException(Exception e)

This should be called on any service exception to convert the exception to an RPCError.

Parameters

e- The exception to convert.

}

---

public *interface* **RPCRequest** [se.natusoft.osgi.aps.api.net.rpc.model] {

This represents a request returned by protocol implementations.

boolean isValid()

Returns true if this request is valid. If this returns false all information except *getError()* is **invalid**, and *getError()* should return a valid *RPCError* object.

RPCError getError()

Returns an *RPCError* object if `isValid() == false`, *null* otherwise.

RPCExceptionConverter getExceptionConverter()

If an exception occurred during the request call, and this returns non null, then the returned converter should be called with the occurred exception to provide an RPCError.

This allows for a specific protocol implementation to handle its own exceptions and provide an appropriate RPCError.

String getServiceQName()

Returns a fully qualified name of service to call. This will be null for protocols where service name is not provided this way. So this cannot be taken for given!

String getMethod()

Returns the method to call. This can return *null* if the method is provided by other means, for example a REST protocol where it will be part of the URL.

boolean hasCallId()

Returns true if there is a call id available in the request.

A call id is something that is received with a request and passed back with the response to the request. Some RPC implementations will require this and some wont.

Object getCallId()

Returns the method call call Id.

A call id is something that is received with a request and passed back with the response to the request. Some RPC implementations will require this and some wont.

RequestIntention getRequestIntention()

Returns the intention of the request.

int getNumberOfParameters()

Return the number of parameters available.

<T> T getIndexedParameter(int index, Class<T> paramClass) throws RequestedParamNotAvailableException

Returns the parameter at the specified index.

Returns

The parameter object or null if indexed parameters cannot be delivered.

Parameters

index- The index of the parameter to get.

paramClass- The expected class of the parameter.

Throws

RequestedParamNotAvailableException- if requested parameter is not available.

}

public *interface* **RPCProtocol** [se.natusoft.osgi.aps.api.net.rpc.service] {

This represents an RPC protocol provider. This API is not enough in itself, it is a common base for different protocols.

String getServiceProtocolName()

Returns

The name of the provided protocol.

String getServiceProtocolVersion()

Returns

The version of the implemented protocol.

String getRequestContentType()

Returns

The expected content type of a request. This should be verified by the transport if it has content type availability.

String getResponseContentType()

Returns

The content type of the response for when such can be provided.

String getRPCProtocolDescription()

Returns

A short description of the provided service. This should be in plain text.

RPCError createRPCError(ErrorType errorType, String message, String optionalData, Throwable cause)

Factory method to create an error object.

Returns

An RPCError implementation or null if not handled by the protocol implementation.

Parameters

errorType- The type of the error.

message- An error messaging.

optionalData- Whatever optional data you want to pass along or null.

cause- The cause of the error.

}

---

public *interface* **StreamedRPCProtocol** extends RPCProtocol

[se.natusoft.osgi.aps.api.net.rpc.service] {

This represents an RPC protocol provider that provide client/service calls with requests read from an InputStream or having parameters passes as strings and responses written to an OutputStream.

HTTP transports can support both *parseRequests(...)* and *parseRequest(...)* while other transports probably can handle only *parseRequests(...)*. **A protocol provider can return null for either of these!** Most protocol providers will support *parseRequests(...)* and some also *parseRequest(...)*.

List<RPCRequest> parseRequests(String serviceQName, Class serviceClass, String method, InputStream requestStream, RequestIntention requestIntention) throws IOException

Parses a request from the provided InputStream and returns 1 or more RPCRequest objects.

Returns

The parsed requests.

Parameters

serviceQName- A fully qualified name to the service to call. This can be null if service name is provided on the stream.

serviceClass- The class of the service to call. Intended for looking for method annotations! Don't try to be "smart" here!

method- The method to call. This can be null if method name is provided on the stream.

requestStream- The stream to parse request from.

requestIntention- The intention of the request (CRUD + UNKNOWN).

Throws

IOException- on IO failure.

RPCRequest parseRequest(String serviceQName, Class serviceClass, String method, Map<String, String> parameters, RequestIntention requestIntention) throws IOException

Provides an RPCRequest based on in-parameters. This variant supports HTTP transports.

Return null for this if the protocol does not support this!

Returns

The parsed requests.

Parameters

serviceQName- A fully qualified name to the service to call. This can be null if service name is provided on the stream.

serviceClass- The class of the service to call. Intended for looking for method annotations! Don't try to be "smart" here!

method- The method to call. This can be null if method name is provided on the stream.

parameters- parameters passed as a

requestIntention- The intention of the request (CRUD + UNKNOWN).

Throws

IOException- on IO failure.

void writeResponse(Object result, RPCRequest request, OutputStream responseStream) throws IOException

Writes a successful response to the specified OutputStream.

Parameters

result- The resulting object of the RPC call or null if void return. If is possible a non void method also returns null!

request- The request this is a response to.

responseStream- The OutputStream to write the response to.

Throws

IOException- on IO failure.

boolean writeErrorResponse(RPCError error, RPCRequest request, OutputStream responseStream) throws IOException

Writes an error response.

Returns

true if this call was handled and an error response was written. It returns false otherwise.

Parameters

error- The error to pass back.

request- The request that this is a response to.

responseStream- The OutputStream to write the response to.

Throws

IOException- on IO failure.

}

APSExtProtocolHTTPTransportProvider

This provides an http transport for simple remote requests to OSGi services that have "APS-Externalizable: true" in their META-INF/MANIFEST.MF. This follows the OSGi extender pattern and makes any registered OSGi services of bundles having the above manifest entry available for remote calls over HTTP. This transport makes use of the aps-external-protocol-extender which exposes services with the above mentioned manifest entry with each service method available as an APSExternallyCallable.The aps-ext-protocol-http-transport-provider for example acts as a mediator between the protocol implementations and aps-external-protocol-extender for requests over HTTP.

Please note that depending on protocol not every service method will be callable. It depends on its arguments and return value. It mostly depends on how well the protocol handles types and can convert between the caller and the service.

This does not provide any protocol, only transport! For services to be able to be called at least one protocol is needed. Protocols are provided by providing an implementation of se.natusoft.osgi.aps.api.net.rpc.service.StreamedRPCProtocol and registering it as an OSGi service. The StreamedRPCProtocol API provides a protocol name and protocol version getter which is used to identify it. A call to an RPC service looks like this:

http://host:port/apsrpc/*protocol*/*version*[/*service*][/*method*]

protocol - This is the name of the protocol to use. An implementation of that protocol must of course be available for this to work. If it isn't you will get a 404 back! The protocol service (RPCProtocol<-StreamedRPCProtocol) provides a name for each protocol. It is this name that is referenced.

version - This is the version of the protocol. If this doesn't match any protocols available you will also get a 404 back.

service - This is the service to call. Depending on the protocol you might not need this. But for protocols that only provide method in the stream data like JSONRPC for example, then this is needed. When provided it has to be a fully qualified service interface class name.

method - This is the method to call. The need for this also depends on the protocol. A REST protocol would need it. The JSONRPC protocol does not. When this is specified in the URL then it will be used even if the protocol provides the method in the request! Please note that a method can be specified on two ways:

• method(type,...)
• method

The method will be resolved in that order. The parameter type specifying version is required when there are several methods with the same name but different parameters. The method name only will give you the last one in that case.

Examples

See examples under the **APSStreamedJSONRPCProtocolProvider** section.

Authentication

Authentication for services are provided in 2 ways. Both require a userid and a password and both validate the user using the APSAuthService.

The 2 alternatives are:

• http://.../apsrpc/**auth:user:password**/protocol/...
• Basic HTTP authentication using header: 'Authorization: Basic {*base 64 encoded user:password*}'.

One of these will be required if the *requireAuthentication* configuration have been enabled.

The help web

Opening the *http://.../apsrpc/_help/* URL will give you a web page that provides a lot of information. This page requires authentication since it register itself with the APSAdminWeb (/apsadminweb) as ”Remote Services” and appears there as a tab, and thus joins in with the APSAdminWeb authentication.

In addition to much of the same information as in this documentation it also lists all protocols tracked by the *APSExternalProtocolExtender* with their name, version, description, and other properties. Next it lists all services that *APSExternalProtocolExtender* provides as callable. Each of these services are a link that can be clicked. Clicking on a service will show all the methods of the service and then list the call url for each method per protocol. Each method listed is also a link, and clicking that link will give you a page where you can provide arguments and then press execute to call the service. The result will be displayed as JSON on the same page. This is very useful for testing and debugging services.

See Also

Also look at the documentation for APSExternalProtocolExtender.

APS RabbitMQ Message Service Provider

This service provides an implementation of APSMessageService 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.

APSMessageService API

Javadoc

APSStreamedJSONRPCProtocolProvider

This provides JSONRPC protocol. It provides both version 1.0 and 2.0 of the protocol. It requires a transport that uses it and services provided by aps-external-protocol-extender to be useful.

The URL format for all of these looks like this:

```
http(s)://host:port/apsrpc/{protocol}/{protocol version}/{service}[/{method}][?params=...]
```

Where:

{protocol} is one of the below described protocols.

{protocol version} is the version of the specified protocol.

{service} is the name of the service to call. Safest is to use a fully qualified name, that is inlcuding package since that will make the service specification unique. It is however possible to only specify the name of the service in which case the first matching will be used.

{method} is the method of the service to call. In the case of JSONREST the method can be skipped and a method will be found based on the HTTP method used to make the call.

JSONRPC version 1.0

This protocol is described at http://json-rpc.org/wiki/specification.

JSONRPC version 2.0

This protocol is describved at http://jsonrpc.org/spec.html.

JSONHTTP version 1.0

This is not any standard protocol at all. It requires both service name and method name on the url, and in case of HTTP GET or DELETE also arguments as ?params=arg:...:arg where values are strings or primitives. For POST, and PUT a JSON array of values need to be written on the stream.

JSONREST version 1.0

This provides a loose API of REST type. It will return HTTP error code on any failure. It has several options for calling. It is possible to specify both service and method to call, but if the method is omitted then a method will be deduced by the HTTP method used.

For HTTP method POST methods starting with one of the following will be matched: *create*, *post*, *new* .

For HTTP method GET methods starting with one of the following will be matched: *read*, *get*.

For HTTP method PUT methods starting with one of the following will be matched: *update*, *put*, *set*, *write*.

For HTTP method DELETE methods starting with one of the following will be matched: *delete*, *remove*.

JSONREST actually extends JSONHTTP and inherits some of its features, like the *params=arg:...:arg* parameter. It however adds an own parameter feature: If a service method takes one MapString, String as parameter, all specified HTTP GET parameters will be provided in this Map.

Also note that for GET and DELETE '...?params=...' must be used to provide parameters to the call, with the above mentioned exception, while for POST and PUT JSON must be provided on the request stream.

Examples

Here is some examples calling services over http with diffent protocols using curl (*requires aps-ext-protocol-http-transport-provider.jar and the called services to be deployed, and specified as externalizable via configuration* (Network/service/external-protocol-extender)):

```
curl --data '{"jsonrpc": "2.0", "method": "getPlatformDescription", "params": [], "id": 1}'
    http://localhost:8080/apsrpc/JSONRPC/2.0/se.natusoft.osgi.aps.api.core.platform.service.APSPlatformService
```

yields

```
{"id": 1, "result": {"description": "My personal development environment.", "type":
    "Development", "identifier": "MyDev"}, "jsonrpc": "2.0"}
```

while

```
curl --get
    http://localhost:8080/apsrpc/JSONHTTP/1.0/se.natusoft.osgi.aps.api.core.platform.service.APSPlatformService/getPla
```

yields

```
{"description": "My personal development environment.", "type": "Development", "identifier":
    "MyDev"}
```

and

```
curl --get
    http://localhost:8080/apsrpc/JSONHTTP/1.0/se.natusoft.osgi.aps.api.misc.session.APSSessionService/createSession\(I
```

yields

```
{"id": "6d25d646-11fc-44c3-b74d-29b3d5c94920", "valid": true}
```

In this case we didn't just use *createSession* as method name, but *createSession(Integer)* though with parentheses escaped to not confuse the shell. This is because there is 2 variants of createSession: createSession(String, Integer) and createSession(Integer). If we don't specify clearly we might get the wrong one and in this case that happens and will fail due to missing second parameter. Also note the *params=5*. On get we cannot pass any data on the stream to the service, we can only pass parameters on the URL which is done by specifying url parameter *params* with a colon (:) separated list of parameters as value. In this case only String and primitives are supported for parameters.

The following is also valid:

```
curl --get
    http://localhost:8080/apsrpc/JSONHTTP/1.0/APSPlatformService/getPlatformDescription
```

Note that this is much shorter. It does not provide a fully qualified name for the APSPlatformService. This is OK. As long as the service name is unique even without package it will be found correctly. The odds of having 2 services with the same name in different packages are quite small so this is rather safe to do.

Note: These examples only works if you have disabled the *requireAuthentication* configuration (network/rpc-http-transport).

See also

Se the documentation for *APSExtProtocolHTTPTransportProvider* for an HTTP transport through which these protocols can be used.

Se the documentation for *APSExternalProtocolExtender* for a description of how services are made available and what services it provides for transport providers.

APSTCPIPService

This service provides, in ways of communication, plain simple TCP/IP communication. Users of this service will however have very little contact with the java.net classes.

The following are the points of this service:

- Simple TCP/IP usage.
- Makes use of an URI to provide what I call a "connection point". tcp:, udp:, and multicast: are supported protocols.

Do note that you do need to have a basic understanding of TCP/IP to use this service!

Security

Makes use of 2 separate services if available for security: *APSTCPSecurityService* and *APSUDPSecurityService*. Neither these nor APSTCPIPService makes any assumptions nor demands on the what and how of the security services implementations. The APSTCPSecurityService must provide secure versions of Socket and ServerSocket, while APSUDPSecureService have 2 methods, one to encrypt the data and one to decrypt the data in a DatagramPacket.

APS currently does not provide any implementation of the APS(TCP/UDP)SecurityService.

Connection Point URIs

The service makes use of URIs to specify where to connect for sending or receiving.

The URI format is this:

protocol://host:port#fragment,fragment

Protocols:

tcp, udp, multicast, named

The *named* protocol just provides a name for the *host* part of the URI. This is not however a hostname! It is a name that has been entered in the service configuration and which has an URI value which will be used instead. So *named://mysvc* will lookup a config value having "mysvc" as destination name and use its destination URI as connection URI. So the valid URI protocols in the configuration is then tcp, udp, and multicast.

Fragments:

secure - If specified then one of the APS(TCP/UDP)SecurityService services will be used.

async (only valid on *tcp* protocol)

Examples

TCP

Write

```
APSTCPIPService tcpipSvc;
...
tcpipSvc.sendStreamedRequest(new URI("tcp://localhost:9999"), new StreamedRequest() {
    void sendRequest(URI connectionPoint, OutputStream requestStream, InputStream responseStream)
    throws IOException {
        // write to requestStream ...

        // read from response stream ...
    }
})
```

Read

```
APSTCPIPService tcpipSvc;
...
tcpipSvc.setStreamedRequestListener(new URI("tcp://localhost:9999"), this);
...
void requestReceived(URI receivePoint, InputStream requestStream, OutputStream responseStream) {
    // Read request from reqStream ...

    // Write response to respStream ...
}
```

Note that there can only be one listener per URI.

UDP / Multicast

Since Multicast uses UDP packets there is no difference between host and port connected UDP or Multicast. The only difference is in the URI where "udp://" is specified for UDP packets and "multicast://" is specified for multicast packets.

Write

```
APSTCPIPService tcpipSvc;
...
bytes[] bytes = "Some data".getBytes();
tcpipSvc.sendDataPacket(new URI("udp://localhost:9999"),  bytes);
```

or

```
tcpipSvc.sendDataPacket(new URI("multicast://all-systems.mcast.net:9999"), bytes);
```

READ

```
APSTCPIPService tcpipSvc;
...
tcpipSvc.addDataPacketListener(new URI("udp://localhost:9999"), this);
...
void dataBlockReceived(URI receivePoint, DatagramPacket packet) {
    byte[] bytes = packet.getData();
    ...
}
```

aps-vertx-event-bus-messaging-provider

This provides an implementation of *APSMessageService* that uses the Vertx event bus for sending and receiving messages on the Vertx cluster. The topics are used as adresses in this case.

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

APSVertxTCPMessagingProvider

This is an implementation of *APSMessageService* that uses Vert.x to provide the TCP communication. It is dependent on the **aps-vertx-provider** bundle which provides reusable *Vertx* instances.

Configuration

This bundle exposes a service interface: `APSVertxTCPMessagingOptions`. This should be implemented and published as an OSGi service. This provider will look for an implementation of that service and configure itself accoding to the supplied configuration.

Here is the API:

```
package se.natusoft.osgi.aps.net.messaging.vertx.api
```

```
/**
 * This is a service API for providers of options to provide an implementation of.
 */
interface APSVertxTCPMessagingOptions {

    /**
     * @return A Map containing Vert.x Net server options. Contents: name, value
     */
    Map<String, Object> getServerOptions()

    /**
     * @return A Map containing Vert.x Net client options. Contents: name, value.
     */
    Map<String, Object> getClientOptions()

    /**
     * @return Mappings between topic and the URI for the topic.
     */
    Map<String, String> getTopicToURIMapping()
}
```

Since this is build on top of Vert.x it takes Vertx options Map:s. See the Vertx documentation for more info.

The topic to URI mapping is a simple mapping per topic what it should listen to for received messages and where it should send published messages.

Here is an example from the test code:

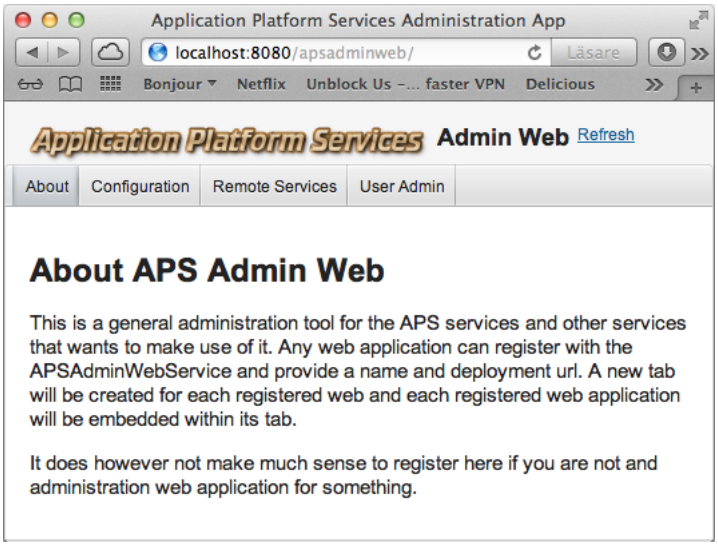
```
Map<String, String> getTopicToURIMapping() {
    return [
            testOne:
    "tcp://localhost:13987?inst=4#in,tcp://localhost:13988/?inst=2#out,tcp://localhost:13989/#out",
            testTwo:
    "tcp://localhost:13988?inst=2#in,tcp://localhost:13987/#out,tcp://localhost:13989/#out",
            testThree:
    "tcp://localhost:13989/#in,tcp://localhost:13988/#out,tcp://localhost:13987/#out"
    ]
}
```

Note that this is Groovy syntax, but it shold be self explanatory what is topic and what is URIs :-).

- There are commas between each URI.
- The protocol is always 'tcp'.
- The fragment part (#) has either an "in" or and "out". This is required. There should always only be one #in, and one or more #out.
- The only query is *inst=n* This is optional.
  - For an #in *inst* number of services listening on host and port of the URI will be created. This is how Vert.x distributes service jobs across processor cores. There should be at least as many as there are cores if you want to utilize the processor to its fullest.
  - For an #out there will be a pool of clients that will hold at most *inst* number of clients. This is for multiple threads to be able to send messages to the same topic. Each client instance will have its own connection to the receiving service. If the pool gets full and no clients are free then a new client will be created, and then deleted again. A warning will be displayed on the log in this case.

As can be seen in the example above, each topic has one service (#in) and 2 clients (#out), one for each of the other 2 topics. So these 3 topics send their messages to the other 2, whatever is sent by one the other 2 will get.

APSAdminWeb



This is a web app for administration of APS. It is really only a shell for different administraion webs. It relys on the *aps-admin-web-service-provider* bundle which publishes the *APSAdminWebService*. Other bundles providing administration web apps register themselves with this service and for each registration APSAdminWeb creates a tab in its gui. Se *APIs* further down for the APSAdminService API. Clicking on ”Refresh” will make APSAdminWeb reload the admin webs registered in *APSAdminWebService*.

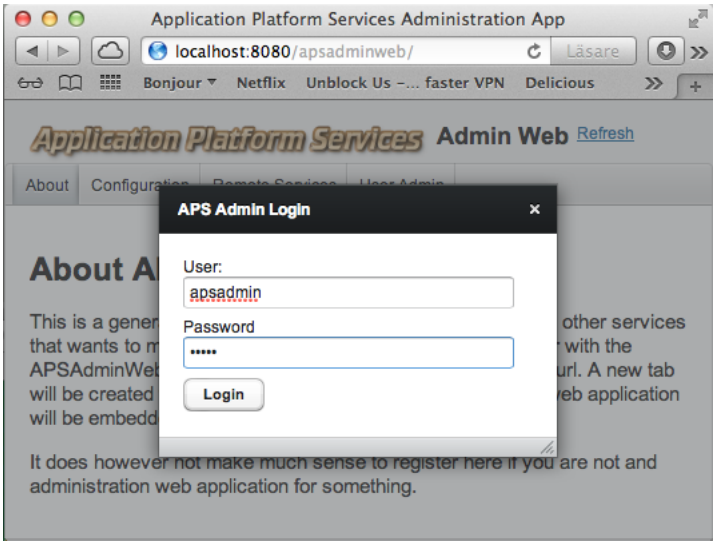
The APSAdminWeb is accessed at **http://host:port/apsadminweb**. What you see there depends on what other admin webs are deployed. Anybody can make an admin web and register it with the *APSAdminWebService*. The admin webs delivered with APS are mainly done using Vaadin. This is in no way a requirement for an admin web. An admin web app can be made in any way what so ever. A side effect of this is that different tabs might have different look and feel. But I choose flexibility over beauty!

The following APS bundles provides a tab in APSAdminWeb:

- *aps-config-admin-web.war* - Allows advanced configuration of bundles/services using APSConfigService.
- *aps-user-admin-web-war* - Administration of users and groups for APSSimpleUserService.
- *aps-ext-protocol-http-transport-provider.war* - Provides a web gui with help for setting up and calling services remotely, and also shows all available services and allows calling them from the web gui for testing/debugging purposes.

Authentication

If *”Configuration tab, Configurations/aps/adminweb/requireauthentication”* property is enabled then the APSAdminWeb requires a login to be accessed. A userid and a password will be asked for. The entered information will be validated by the APSAuthService. The *aps-simple-user-service-auth-service-provider.jar* bundle provides an implementation of this service that uses the *APSSimpleUserService* service. The APSAuthService is however simple enough to implement yourself to provide login to whatever you want/need.



Making an admin web participating in the APSAdminWeb login.

There is an APSSessionService that was made just for handling this. It is not a HTTP session, just a service hanling sessions. It is provided by the *aps-session-service-provider.jar* bundle. When a session is created you get a session id (an UUID) that needs to be passed along to the other admin webs through a cookie. *APSWebTools* (*aps-web-tools.jar* (not a bundle!)) provides the APSAdminWebLoginHandler class implementing the LoginHandler interface and handles all this for you.

You need to provide it with a BundleContext on creation since it will be calling both the *APSAuthService* and *APSSessionService*:

```
this.loginHandler = new APSAdminWebLoginHandler(bundleContext);
```

Then to validate that there is a valid login do:

```
this.loginHandler.setSessionIdFromRequestCookie(request);
if (this.loginHandler.hasValidLogin()) {
    ...
}
else {
    ...
}
```

APSAdminWebService APIs

public *interface* **APSAdminWebService** [se.natusoft.osgi.aps.apsadminweb.service] {

This service registers other specific administration web applications to make them available under a common administration gui.

public void registerAdminWeb(AdminWebReg adminWebReg) throws IllegalArgumentException

Registers an admin web application.

Parameters

adminWebReg- Registration information for the admin web.

Throws

IllegalArgumentException- if the admin web has already been registered or if it is using the

public void unregisterAdminWeb(AdminWebReg adminWebReg)

Unregisters a previously registered admin web. This is failsafe. If it has not been registered nothing happens.

Parameters

adminWebReg- Registration information for the admin web. Use the same as registered with.

public List<AdminWebReg> getRegisteredAdminWebs()

Returns

All currently registered admin webs.

}

---

public *class* **AdminWebReg** [se.natusoft.osgi.aps.apsadminweb.service.model] {

This model holds information about a registered admin web application.

public AdminWebReg(String name, String version, String description, String url)

Creates a new AdminWebReg instance.

Parameters

name- A (short) name of the admin web.

version- The version of the admin web.

description- A longer description of the admin web.

url- The deployment url of the admin web.

public String getName()

Returns

The (short) name of the admin web.

public String getVersion()

Returns

The version of the admin web.

public String getDescription()

Returns

The description of the admin web.

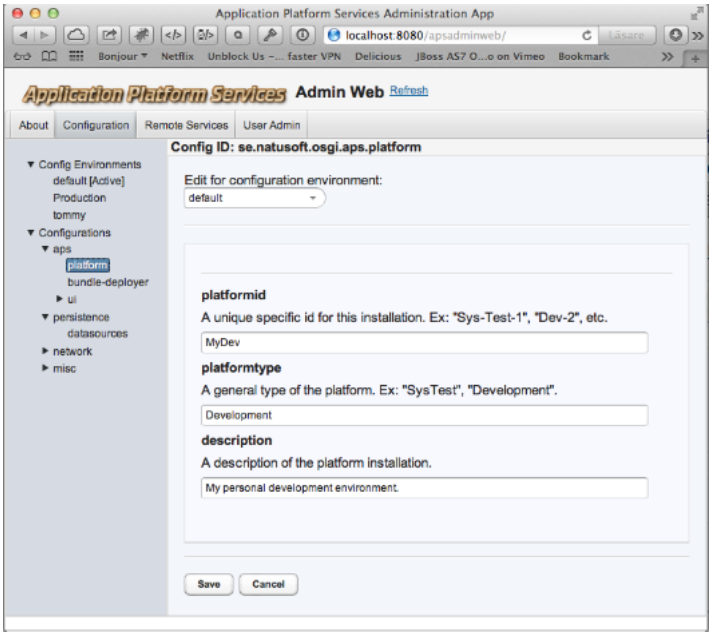
public String getUrl()

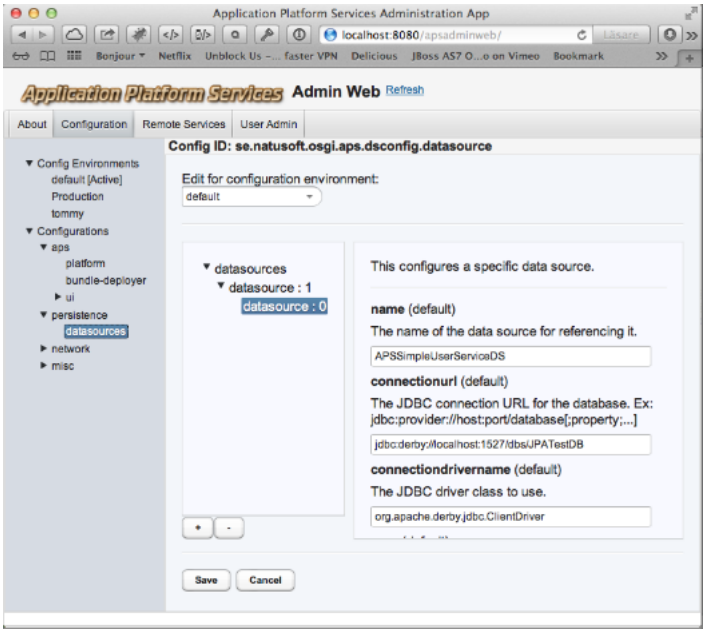
Returns

The deployment url of the admin web.

}

APSConfigAdminWeb





This allows editing configurations registered with the *APSConfigService*. Configurations are only available in the APSConfigAdminWeb while the bundle providing the configuration model are deployed. The actual saved configurations live on disk and remains after a bundle is stopped. It will be available again when the bundle is started again. But the bundle have to be running and regisitering its configuration with the *APSConfigService* for them to be editable in this admin app!

As can be seen in the screenshots above it provides a simpler gui for simple configs, and a more advanced gui for structured configurations containing list of other configuration models.

Config Environments

Under this node all available configuration environments are listed. Right clicking on the node will drop down a menu alternative to create a new configuration environment. Right clicking on a configuration environment pops up a menu that allows it to be set as active configuration environment or to delete the configuration environment. Just clicking on a configuration environment

allows it to be edited on the right side. The active configuration environment cannot however be edited, only viewed.

Configurations

This tree cannot be edited. What is here is the configurations registered by bundles. They can be selected to edit the selected configuration to the right. The screenshots above shows 2 examples of such. Please note that the screenshots were taken on a Mac with Mountain Lion and thus does not show scrollbars unless scrolling. The right side of the second screenshot where things are slightly cutoff at the bottom are scrollable!

On top of the right side box there is a dropdown menu that shows/selects the configuration environment you are editing configuration values for. Only configuration values that are marked in the configuration model as being configuration environment specific will get different values per configuration environement. Those values that are configuration environment specific are identified by having the configuration environment in parentesis after the configuration value key. If you switch the configuration environment in the top dropdown menu you will se that these values change.

Boolean configuration values will be shown as checkboxes. Date configuration values will have a date field where the user can write a date or click the button on the end to bring upp a calendar to select from. Date configuration values can also specify the date format (as described here) in the configuration model. This is used to display the date in the field and parse any entered date. So different date fields can have different formats!

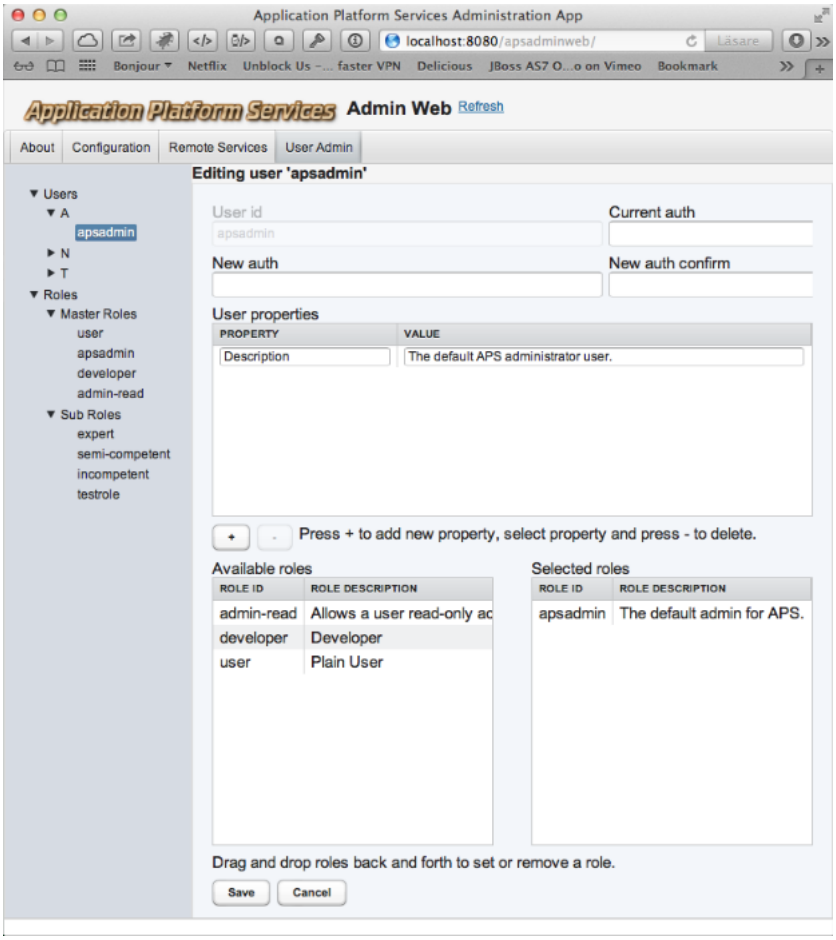
The configuration models are annotated and provide descriptions of the values which are shown in the gui to make it easy for the person doing the configuration to know what the configuration is about.

As soon as the configuration changes are saved they become active. The code using the configurations doesn’t need to do anything. The next reference to a configuration value will return the new value.

See also

Also se the APSConfigService documentation.

APSUserAdminWeb



APSUserAdminWeb provides user and group administration for the *APSSimpleUserService*.

Users are splitt into groups of the first character in the userid to make them a little bit easier to find if there are many. So all userids starting with ’a’ or ’A’ will be under Users/A and so on.

Right click on the *Users* node to create a new user.

Right click on the *Roles* node to create a new role.

Warning: For the roles it is fully possible to create circular dependencies! **Dont!** (There is room for improvement on this point!)

There is not anything more to say about this. It should be selfexplanatory!

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 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.
```

Day Specification version License

Day Management AG ("Licensor") is willing to license this specification to you ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS LICENSE AGREEMENT ("Agreement"). Please read the terms and conditions of this Agreement carefully.

Content Repository for JavaTM Technology API Specification ("Specification") Version: 2.0 Status: FCS Release: 10 August 2009

Copyright 2009 Day Management AG Barfüsserplatz 6, 4001 Basel, Switzerland. All rights reserved.

NOTICE; LIMITED LICENSE GRANTS

1. License for Purposes of Evaluation and Developing Applications.

Licensor hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under Licensor's applicable intellectual property rights to view, download, use and reproduce the Specification only for the purpose of internal evaluation. This includes developing applications intended to run on an implementation of the Specification provided that such applications do not themselves implement any portion(s) of the Specification.

1. License for the Distribution of Compliant Implementations. Licensor also grants you a perpetual,

non-exclusive, non-transferable, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights or, subject to the provisions of subsection 4 below, patent rights it may have covering the Specification to create and/or distribute an Independent Implementation of the Specification that: (a) fully implements the Specification including all its required interfaces and functionality; (b) does not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Specification or Specifications being implemented; and (c) passes the Technology Compatibility Kit (including satisfying the requirements of the applicable TCK Users Guide) for such Specification ("Compliant Implementation"). In addition, the foregoing license is expressly conditioned on your not acting outside its scope. No license is granted hereunder for any other purpose (including, for example, modifying the Specification, other than to the extent of your fair use rights, or distributing the Specification to third parties).

2. Pass-through Conditions. You need not include limitations (a)-(c) from the previous paragraph or any other particular "pass through" requirements in any license You grant concerning the use of your Independent Implementation or products derived from it. However, except with respect to Independent Implementations (and products derived from them) that satisfy limitations (a)-(c) from the previous paragraph, You may neither: (a) grant or otherwise pass through to your licensees any licenses under Licensor's applicable intellectual property rights; nor (b) authorize your licensees to make any claims concerning their implementation's compliance with the Specification.
3. Reciprocity Concerning Patent Licenses. With respect to any patent claims covered by the license granted under subparagraph 2 above that would be infringed by all technically feasible implementations of the Specification, such license is conditioned upon your offering on fair, reasonable and non-discriminatory terms, to any party seeking it from You, a perpetual, non-exclusive, non-transferable, worldwide license under Your patent rights that are or would be infringed by all technically feasible implementations of the Specification to develop, distribute and use a Compliant Implementation.
4. Definitions. For the purposes of this Agreement: "Independent Implementation" shall mean an implementation of the Specification that neither derives from any of Licensor's source code or binary code materials nor, except with an appropriate and separate license from Licensor, includes any of Licensor's source code or binary code materials; "Licensor Name Space" shall mean the public class or interface declarations whose names begin with "java", "javax", "javax.jcr" or their equivalents in any subsequent naming convention adopted by Licensor through the Java Community Process, or any recognized successors or replacements thereof; and "Technology Compatibility Kit" or "TCK" shall mean the test suite and accompanying TCK User's Guide provided by Licensor which corresponds to the particular version of the Specification being tested.
5. Termination. This Agreement will terminate immediately without notice from Licensor if you fail to comply with any material provision of or act outside the scope of the licenses granted above.
6. Trademarks. No right, title, or interest in or to any trademarks, service marks, or trade names of Licensor is granted hereunder. Java is a registered trademark of Sun Microsystems, Inc. in the United States and other countries.
7. Disclaimer of Warranties. The Specification is provided "AS IS". LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT (INCLUDING AS A CONSEQUENCE OF ANY PRACTICE OR IMPLEMENTATION OF THE SPECIFICATION), OR THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE. This document does not represent any commitment to release or implement any portion of the Specification in any product.

The Specification could include technical inaccuracies or typographical errors. Changes are periodically added to the information therein; these changes will be incorporated into new versions of the Specification, if any. Licensor may make improvements and/or changes to the product(s) and/or the program(s) described in the Specification at any time. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

1. Limitation of Liability. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL,

INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

2. Report. If you provide Licensor with any comments or suggestions in connection with your use of the Specification ("Feedback"), you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Licensor a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

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.

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 AGR

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 Cont
"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 displa
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, imp
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 o
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
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 Prog
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, o

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 WARRANT

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
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 softw
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
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
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 mo
```

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.