

Application Platform Services

User Guide

Version: 0.9.0

Author: Tommy Svensson

Copyright © 2013 Natusoft AB

Table of Contents

1 Application Platform Services (APS)	1
1.1 Features	1
1.1.1 Current	1
1.1.2 Planned	1
1.1.3 Ideas	2
1.2 Pre Setup	2
1.3 Javadoc	2
2 APSConfigService	3
2.1 Configuration Environments	3
2.2 Making a config class	3
2.2.1 The config values	4
2.2.2 The config annotations	4
2.2.2.1 @APSConfigDescription	4
2.2.2.2 @APSConfigItemDescription	4
2.2.2.3 @APSDefaultValue	5
2.2.3 Auto managed configurations	5
2.2.3.1 Variant 1: A simple non instantiated static member of config model type	6
2.2.3.2 Variant 2: A static instantiated ManagedConfig<ConfigModel> member.	6
2.3 API Usages	7
2.3.1 The configuration service usage	7
2.3.2 The configuration admin service usage	7
2.4 The complete APS API	7
2.5 A word of advice	7
2.6 APSConfigAdminWeb screenshots	7
3 APSFilesystemService	10
3.1 Setup	10
3.2 The service	10
3.3 The APIs for this service	11
4 APSPlatformService	20
4.1 APIs	20
5 APSJSONLib	22
5.1 APIs	22
6 APSToolsLib	42
6.1 APSServiceTracker	42
6.1.1 Services and active service	42
6.1.2 Providing a logger	42
6.1.3 Tracker as a wrapped service	43
6.1.4 Using the tracker in a similar way to the OSGi standard tracker	43
6.1.5 Accessing a service by tracker callback	43
6.1.5.1 onServiceAvailable	43
6.1.5.2 onServiceLeaving	43
6.1.5.3 onActiveServiceAvailable	44
6.1.5.4 onActiveServiceLeaving	44
6.1.5.5 withService	44
6.1.5.6 withServiceIfAvailable	44
6.1.5.7 withAllAvailableServices	44
6.2 APSLogger	44
6.3 APSContextWrapper	45
6.4 ID generators	45
7 APSWebTools	47
7.1 APIs	47

8 APSAuthService	52
8.1 APSSimpleUserServiceAuthServiceProvider	52
8.2 API	52
9 APSSimpleUserService	55
9.1 Basic example	55
9.2 Setup	55
9.3 APIs	57
10 APSDataSource	68
10.1 APIs	68
11 APSJPAService	70
11.1 APIs	70
12 APSJSONService	73
13 APSResolvingBundleDeployer	74
13.1 Configuration	74
14 APSSessionService	75
14.1 APIs	75
15 APSDiscoveryService	78
15.1 APIs	78
16 APSExternalProtocolExtender	82
16.1 The overall structure	82
16.2 APSExternalProtocolService	83
16.2.1 Protocols	83
16.2.2 Getting information about services and protocols.	83
16.3 See also	83
16.4 APIs	83
17 APSExtProtocolHTTPTransportProvider	97
17.1 Example	97
17.2 Authentication	98
17.3 The help web	98
17.4 See Also	98
18 APSGroups	99
18.1 OSGi service usage	99
18.1.1 Getting the service	99
18.1.2 Joining a group	99
18.1.3 Sending a message	99
18.1.4 Receiving a message	99
18.1.5 Leaving a group	100
18.2 Library usage	100
18.2.1 Setting up	100
18.2.2 Joining a group	100
18.2.3 Sending and receiving messages	100
18.2.4 Leaving a group	100
18.2.5 Shutting down	100
18.3 Net time	100
18.4 Configuration	101
18.4.1 OSGi service	101
18.4.2 Library	101
18.5 APIs	102
19 APSStreamedJSONRPCProtocolProvider	107
19.1 See also	107

1 Application Platform Services (APS)

OSGi Application Platform Services - A "smorgasbord" of OSGi services that focuses on ease of use and good enough functionality for many but won't fit all. It can be seen as osgi-ee-light-and-easy. The services are of platform type: configuration, database, JPA, content, etc.

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

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

Another point of APS is to be OSGi server independent.

APS is made using basic OSGi functionality and is not using blueprint and other fancy stuff (of which I'm not a believer, I like to be in full control :-)).

1.1 Features

1.1.1 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 its admin web.
- A filesystem service that provides a persistent filesystem outside of the OSGi server. The configuration service makes use of this to store configurations.
- 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 and 2.0 protocols, and an http transport. Protocols have two defined service APIs whose implementations can just be dropped in to make them available. The http transport can make use of any deployed protocol.
- A multicast discovery service.
- A session service (not http!). This is used by apsdadminweb 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 .../apsdadminweb admin gui.
- A user service. Provides basic user management including roles/groups. Is accompanied with a admin GUI (plugged into apsdadminweb) for administration of 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.

1.1.2 Planned

- A log service with a log viewer GUI. The GUI should support server push and also allow for filtering of logs and configuration of what logs go to what log files.

- A REST/JSON protocol for use with aps-external-protocol-extender.
- Anything else relevant I come up with and consider fun to do :-).

1.1.3 Ideas

- JDBC connection pool service (based on some open source connection pool implementation). Will use the Data source service to create connection pools.
- Since JBoss is apparently having trouble getting WABs to work (they are still using PAX, but claim that they have solved this in 7.2 that will not build when checked out from GitHub and don't seem to be released anytime soon) I am considering to add support for their WAR->OSGi service bridge though I haven't had much luck in getting that to work either so far.
- 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).

1.2 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 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 will work but is not optimal!

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

1.3 Javadoc

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

2 APSConfigService

This is not the simple standard OSGi service configurations, but more an application config that can also be used for services. It supports structured configurations including lists of items and lists of subconfigurations. Code that uses the configuration provide one or more configuration classes with config items. These are registered with the config service, which makes them editable/publishable though and admin web app. After registration an instance of the config can be gotten containing published or default values. Alternatively the config class is specified with a fully qualified name in the *APS-Configs*: MANIFEST.MF entry. In this case the configuration service acts as an extender and automatically registers and provides an instance of the config for you, without having to call the config service.

2.1 Configuration Environments

The APSConfigService supports different configuration environments. The idea is to define one config environment per installation. Configuration values can either be configuration environment specific or the same for all environments. See @ConfigItemDescription below for more information on specifying configuration environment specific values.

2.2 Making a config class

Here is an example:

```
@APSConfigDescription(
    version="1.0",
    configId="se.natusoft.aps.example.myconfig",
    group="examples",
    description="An example configuration model"
)
public class MyConfig extends APSConfig {

    @APSConfigItemDescription(
        description="Example of simple value."
    )
    public APSConfigValue simpleValue;

    @APSConfigItemDescription(
        description="Example of list value."
    )
    public APSConfigValueList listValue;

    @APSConfigItemDescription(
        description="One instance of MySubConfig model."
    )
    public MySubConfig mySubConfig;

    @APSConfigItemDescription(
        description="Multiple instances of MySubConfig model."
    )
    public APSConfigList<MySubConfig> listOfMySubConfigs;

    @APSConfigDescription(
        version="1.0",
        configId="se.natusoft.aps.example.myconfig.mysubconfig",
        description="Example of a subconfig model. Does not have to be inner class!"
    )
    public static class MySubConfig extends APSConfig {

        @APSConfigItemDescription(
            description="Description of values."
        )
        public APSConfigValueList listOfValues;

        @APSConfigItemDescription(
            description="Description of another value."
        )
    }
}
```

```

    public APSConfigValue anotherValue;
}
}

```

2.2.1 The config values

Now you might be wondering, why not an interface, and why *public* and why *APSConfigValue*, *APSConfigValueList*, and *APSConfigList*?

The reason for not using an interface and provide a `java.lang.reflect.Proxy` implementation of it is that OSGi has separate class loaders for each bundle. This means a service cannot proxy an interface provided by another bundle. Well, there are ways to go around that, but I did not want to do that unless that was the only option available. In this case it wasn't. Therefore I use the above listed APS*Value classes as value containers. They are public so that they can be accessed and set by the APSConfigService. When you get the main config class instance back from the service all values will have valid instances. Each APS*Value has an internal reference to its config value in the internal config store. So if the value is updated this will be immediately reflected since it is referencing the one and only instance of it in the config store.

All config values are strings! All config values are stored as strings. The **APSConfigValue** container however have *toBoolean()*, *toDate()*, *toDouble()*, *toFloat()*, *toInt()*, *toLong()*, *toByte()*, *toShort()*, and *toString()* methods on it.

The **APSConfigList<Type>** container is an *java.lang.Iterable* of <Type> type objects. The <Type> cannot however be anything. When used directly in a config model it must be <Type extends APSConfig>. That is, you can only specify other config models extending APSConfig. The only exception to that is **APSConfigValueList** which is defined as:

```
public interface APSConfigValueList extends APSConfigList<APSConfigValue> {}
```

- Use **APSConfigValue** for plain values.
- Use **APSConfigValueList** for a list of plain values.
- Use **MyConfigModel extends APSConfig** for a subconfig model.
- Use **APSConfigList<MyConfigModel extends APSConfig>** for a list of subconfig models.

2.2.2 The config annotations

The following 3 annotations are available for use on configuration models.

2.2.2.1 @APSConfigDescription

```

@APSConfigDescription(
    version="1.0",
    configId="se.natusoft.aps.example.myconfig",
    group="docs.examples",
    description="An example configuration model"
)

```

This is an annotation for a configuration model.

version - The version of the config model. This is required.

configId - The unique id of the configuration model. Use same approach as for packages. This is required.

group - This specifies a group or rather a tree branch that the config belongs under. This is only used by the configuration admin web app to render a tree of configuration models. This is optional.

description - This describes the configuration model.

2.2.2.2 @APSConfigItemDescription

```
@APSConfigItemDescription(
    description="Example of simple value.",
    datePattern="yyMMdd",
    environmentSpecific=true/false,
    isBoolean=true/false,
    validValues={"high", "medium", "low"},
)
```

This is an annotation for a configuration item within a configuration model.

description - This describes the configuration value. The configuration admin web app uses this to explain the configuration value to the person editing the configuration. This is required.

datePattern - This is a date pattern that will be passed to SimpleDateFormat to convert the date in the string value to a java.util.Date object and is used by the `toDate()` method of APSConfigValue. This date format will also be displayed in the configuration admin web app to hint at the date format to the person editing the configuration. The configuration admin web app will also use a calendar field if this is available. The calendar field has a complete calendar popup that lets you choose a date. This is optional.

environmentSpecific - This indicates that the config value can have different values depending on which config environment is active. This defaults to false in which case the value will apply to all config environments. This is optional.

isBoolean - This indicates that the config value is of boolean type. This is used by the configuration admin web app to turn this into a checkbox rather than a text field. This defaults to false and is this optional.

validValues - This is an array of strings ({ "...", ..., "..."}) containing the only valid values for this config value. This is used by the configuration admin web app to provide a dropdown menu of the alternatives rather than a text field. This defaults to {} and is thus optional.

defaultValue - This is an array of @APSDDefaultValue annotations. See the description of this annotation below. This allows not only for providing a default value, but for providing a default value per config environment (which is why there is an array of @APSDDefaultValue annotations!). Thus you can deliver pre configured configuration for all configuration environments. If a config environment is not specified for a default value then it applies for all configuration environments. Some configuration values are better off without default values, like hosts and ports for other remote services. The application/server maintenance people responsible for an installation in general knows this information better than the developers.

2.2.2.3 @APSDDefaultValue

```
@APSDDefaultValue {
    configEnv="production",
    value="15"
}
```

configEnv - This specifies the configuration environment this default value applies to. "default" means all/any configuration environment and is the default value if not specified.

value - This is the default value of the configuration value for the configuration environment specified by configEnv.

2.2.3 Auto managed configurations

It is possible to let the APSConfigService act as an extender and automatically register and setup config instances on bundle deploy by adding the **APS-Configs: MANIFEST.MF** header and a comma separated list of fully qualified names of config models. There are two variants of how to define the auto managed instance.

Warning: Auto managed configurations cannot ever be accessed during bundle activation in default activation thread! If the activation code starts a new thread then it is OK to access auto managed configuration in that thread, but only with variant 2! (the thread have to put itself to sleep until the configuration becomes managed. This is described below).

2.2.3.1 Variant 1: A simple non instantiated static member of config model type

Example:

```
@APSCfgDescription(
    version="1.0",
    configId="se.natusoft.aps.example.myconfig",
    group="examples",
    description="An example configuration model"
)
public class MyConfig extends APSConfig {

->  public static MyConfig myConfig;  <-

    @APSCfgItemDescription(
        description="Example of simple value.",
    )
    public APSConfigValue simpleValue;

    @APSCfgItemDescription(
        description="Example of list value."
    )
    public APSConfigValueList listValue;
    ...
}
```

To access this variant of managed config do:

```
MyConfig.myConfig.simpleValue.toString()/toInt()/toDouble()/...
```

A warning: This variant does not provide any support for determining if the configuration has become managed yet. If you access it too early it will be null. Therefore you should only use this variant if you know it will become managed before it is referenced. The other variant allows you to check and wait for a config to become managed.

2.2.3.2 Variant 2: A static instantiated ManagedConfig<ConfigModel> member.

Example:

```
@APSCfgDescription(
    version="1.0",
    configId="se.natusoft.aps.example.myconfig",
    group="examples",
    description="An example configuration model"
)
public class MyConfig extends APSConfig {

->  public static final ManagedConfig<MyConfig> managed = new
ManagedConfig<MyConfig>();  <-

    @APSCfgItemDescription(
        description="Example of simple value.",
    )
    public APSConfigValue simpleValue;

    @APSCfgItemDescription(
        description="Example of list value."
    )
    public APSConfigValueList listValue;
    ...
}
```

There is a possibility that code started in a bundle, especially threads might start running before the config has become managed. In such cases the following will solve that:

```
if (!MyConfig.managed.isManaged()) {
    MyConfig.managed.waitUntilManaged();
}
```

Do not ever do this during start() of a Bundle activator! That would cause a never ending dead-lock!

To access this variant of managed config do:

```
MyConfig.managed.get().simpleValue.toString()/toInt()/toDouble()/...
```

2.3 API Usages

2.3.1 The configuration service usage

The APSConfigService API looks like this:

```
public interface APSConfigService {
    void registerConfiguration(Class<? extends APSConfig> configClass, boolean
forService) throws APSConfigException;
    void unregisterConfiguration(Class<? extends APSConfig> configClass);
    <Config extends APSConfig> Config getConfiguration(Class<Config> configClass)
throws APSConfigException;
}
```

On bundle start you register the configuration. On bundle stop you unregister it. Inbetween you access it. It is a good idea to call getConfiguration(...) after register on bundle start and the pass this instance to your services, etc.

If the *forServices* flag is *true* then this configuration will also be registered in the standard OSGi configuration service. Please be warned however that APSConfigService stores its configuration values in properties files, but with rather complex keys. For non structured, flat configurations it might make some sense to register it with the standard osgi service also, but in most cases there is no point in doing this. I'm not even sure why I have this option!

Please note that if you are using managed configs (see above) then you never need to call this service API, not even lookup/track the APSConfigService!

2.3.2 The configuration admin service usage

The APSconfigAdminService only needs to be used if you implement a configuration editor. APSConfigAdminWeb uses this API for example. See the javadoc for the API.

2.4 The complete APS API

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

2.5 A word of advice

It is quite possible to make config structures of great complexity. **DON'T!** Even if it seems manageable from a code perspective it might not be that from a admin perspective. Keep it simple always apply!

2.6 APSConfigAdminWeb screenshots

Application Platform Services Admin Web [Refresh](#)

About Configuration Remote Services User Admin

Editing config environment 'default'

- ▼ Config Environments
 - default [Active]
 - Production
 - tommy
- ▼ Configurations
 - aps
 - persistence
 - network
 - misc

Config environment name

default

Description of config environment.

This is created when env is asked for and none have been created!

Save Cancel

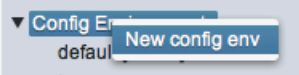
Application Platform Services Admin Web [Refresh](#)

About Configuration Remote Services User Admin

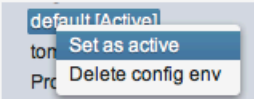
Configuration Environments

There are usually several deployment environments where some configurations do differ per environment. The APS configurations supports different values for different environments. This is the place where you define environments. A common scenario is *development*, *systemtest*, *acceptancetest*, and *production*. In some cases there might be several of each. Example: *syst1*, *syst2*, *syst3*.

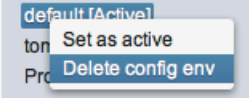
To create a new configuration environment right-click on "Config Environments" and select "New config env".



To select one configuration environment as active right-click on it in the menu and select "Set as active".



To delete a configuration environment right-click on it in the menu and select "Delete config env".



Application Platform Services Admin Web [Refresh](#)

AboutConfigurationUser AdminRemote Services

► Config Environments

▼ Configurations

► aps

► persistence

▼ network

► service

rpc-http-transport

discovery

groups

► misc

Config ID: se.natusoft.osgi.aps.groups

Edit for configuration environment:

default

multicastaddress
The multicast address to use.

224.0.0.1

multicastport
The multicast target port to use.

58100

sendtimeout
The number of seconds to allow for a send of a message before timeout.

120

resendinterval
The number of seconds to wait before a packet is resent if not acknowledged. sendTimeout / resendInterval = the number or resends before giving up.

5

memberannounceinterval
The interval in seconds that members announce that they are (still) members. If a member has not announced itself again within this time other members of the group will drop the member.

10

Save

Cancel

Application Platform Services Admin Web [Refresh](#)

About Configuration Remote Services User Admin

Config ID: se.natusoft.osgi.aps.dsconfig.datasource

Edit for configuration environment:
default

▼ Config Environments
▼ Configurations
 ▶ aps
 ▼ persistence
 datasources
 ▶ network
 ▶ misc

▼ datasources
 ▼ datasource : 1
 datasource : 0

+

-

Save Cancel

This configures a specific data source.

name (default)
The name of the data source for referencing it.
APSSimpleUserServiceDS

connectionurl (default)
The JDBC connection URL for the database. Ex: jdbc:provider://host:port/database[;property;...]
jdbc:derby://localhost:1527/dbs/JPATestDB

connectiondrivername (default)
The JDBC driver class to use.
org.apache.derby.jdbc.ClientDriver

user (default)
The database user to login with.
derby

3 APSFilesystemService

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

The `APSConfigService` uses the `APSFilesystemService` to store its configurations.

3.1 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.

3.2 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.

3.3 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 message 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 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 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 APSFileImpl.

}

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

This represents an APSFilesystemService filesytem.

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 specifeid 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;
APSFilesystemImpl fs;

if (fss.hasFilesystem("my.file.system")) {
    fs = fss.getFilesystem("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.*

}

4 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*.

4.1 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.

}

5 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.

This does not try to be an alternative to Jackson! This is used internally by other services.

5.1 APIs

```
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 IOException
```

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

IOException - on failure.

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

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

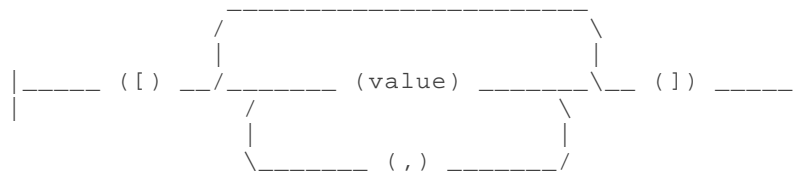
IOException

}

public class JSONArray extends JSONValue [se.natusoft.osgi.aps.json] {

This class is based on the structure defined on <http://www.json.org/>.

This represents the "array" diagram on the above mentioned web page:



@author Tommy Svensson

public JSONArray()

Creates a new JSONArray for writing JSON output.

public JSONArray(JSONErrorHandler errorHandler)

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

Parameters

errorHandler

public void addValue(JSONValue value)

Adds a value to the array.

Parameters

value - The value to add.

public List<JSONValue> getAsList()

Returns the array values as a List.

public <T extends 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 class JSONBoolean extends JSONValue [se.natusoft.osgi.aps.json] {

This class is based on the structure defined on <http://www.json.org/>. @author Tommy Svensson

public JSONBoolean(boolean value)

Creates a new JSONBoolean instance for writing JSON output.

Parameters

value - The value for this boolean.

public JSONBoolean(JSONErrorHandler errorHandler)

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

Parameters

errorHandler

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

This is called on warnings or failures. @author Tommy Svensson

void warning(String message)

Warns about something.

Parameters

message - The warning message.

void fail(String message, Throwable cause) throws RuntimeException

Indicate failure.

Parameters

message - The failure message.

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

Throws

RuntimeException - This method must throw a RuntimeException.

}

public class JSONNull extends JSONValue [se.natusoft.osgi.aps.json] {

This class is based on the structure defined on <http://www.json.org/>. @author Tommy Svensson

public JSONNull()

Creates a new JSONNull instance for writing JSON output.

public JSONNull(JSONErrorHandler errorHandler)

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

Parameters

errorHandler

public String toString()**Returns**

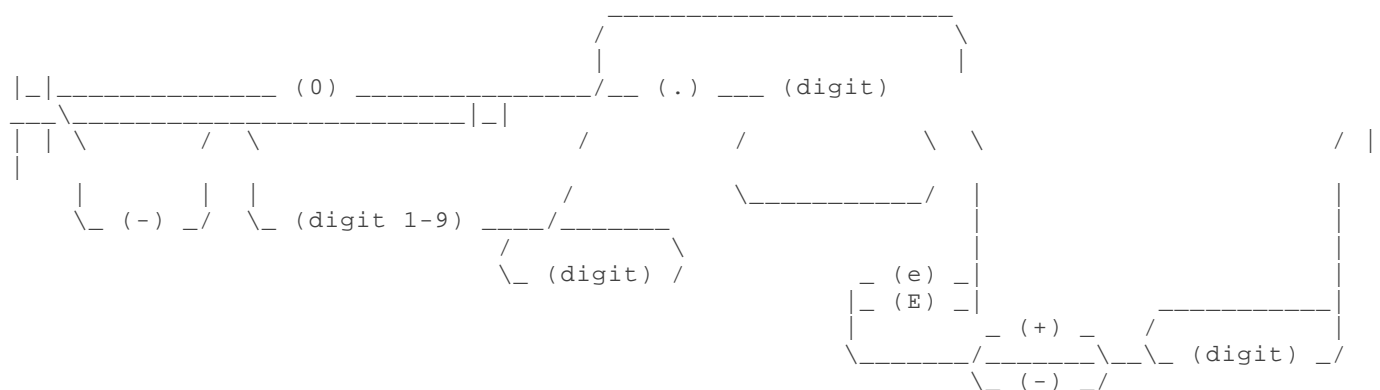
as String.

}

public class JSONNumber extends JSONValue [se.natusoft.osgi.aps.json] {

This class is based on the structure defined on <http://www.json.org/>.

This represents the "number" diagram on the above mentioned web page:



@author Tommy Svesson

public JSONNumber(Number value)

Creates a new JSONNumber instance for writing JSON output.

Parameters

value - The numeric value.

public JSONNumber(JSONErrorHandler errorHandler)

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

Parameters

errorHandler - The error handle to use.

public Number toNumber()

Returns the number as a Number.

public double toDouble()

Returns the number as a double value.

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

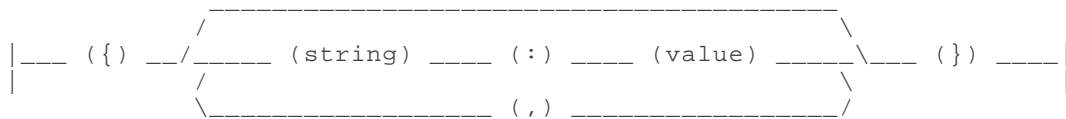
type - The type of the returned number.

}

```
public class JSONObject extends JSONValue [se.natusoft.osgi.aps.json] {
```

This class is based on the structure defined on <http://www.json.org/>.

It represents the "object" diagram on the above mentioned web page:



This is also the starting point.

To write JSON, create a new JSONObject (new JSONObject()) and call addProperty(name, value) for children. Then do jsonObj.writeJSON(outputStream).

To read JSON, create a new JSONObject (new JSONObject(jsonErrorHandler)) and then do jsonObj.readJSON(inputStream). Then use getProperty(name) to extract children. @author Tommy Svensson

```
public JSONObject() } /** * Creates a new JSONObject instance for reading JSON input or writing JSON
output. * * errorHandler */ public JSONObject(JSONErrorHandler errorHandler)
```

Creates a JSONObject instance for writing JSON output.

```
public Set<JSONString> getPropertyNames()
```

Returns the names of the available properties.

```
public JSONValue getProperty(JSONString name)
```

Returns the named property.

Parameters

name - The name of the property to get.

```
public JSONValue getProperty(String name)
```

Returns the named property.

Parameters

name - The name of the property to get.

public void addProperty(JSONString name, JSONValue value)

Adds a property to this JSONObject instance.

Parameters

name - The name of the property.

value - The property value.

```
public void addProperty(String name, JSONValue value)
```

Adds a property to this JSONObject instance.

Parameters

name - The name of the property.

value - The property value.

}

```
public class JSONString extends JSONValue [se.natusoft.osgi.aps.json] {
```

This class is based on the structure defined on <http://www.json.org/>.

This represents the "string" diagram on the above mentioned web page:

```
|____ (") -|____|____ (Any UNICODE character except " or \ or control character) ____|____|
|"") ____|
|
|      |
|      |_____ /
|      |
|      |_____ (\) ____ (") (quotation mark) _____|
|      |_____ (\) (reverse solidus) _____|
|      |_____ (/) (solidus) _____|
|      |_____ (b) (backspace) _____|
|      |_____ (f) (formfeed) _____|
|      |_____ (n) (newline) _____|
|      |_____ (r) (carriage return) _____|
|      |_____ (t) (orizontal tab) _____|
|      |_____ (u) (4 hexadecimal digits) _____|
```

@author Tommy Svensson

public JSONString(String value)

Creates a new JSONString for writing JSON output.

Parameters

value - The value of this JSONString.

public JSONString(JSONErrorHandler errorHandler)

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

Parameters

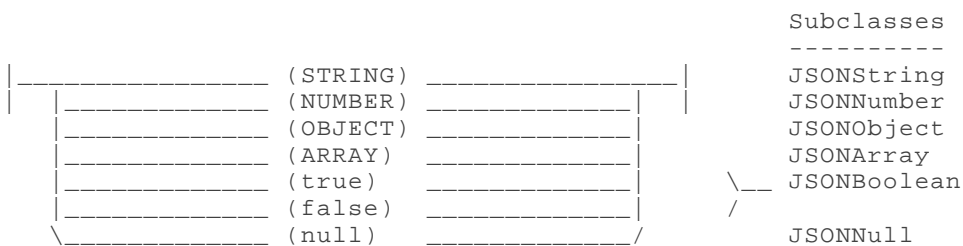
errorHandler

}

public abstract class JSONValue [se.natusoft.osgi.aps.json] {

This class is based on the structure defined on <http://www.json.org/>.

This is a base class for all other JSON classes. It represents the "value" diagram on the above mentioned web page:*



@author Tommy Svensson

protected JSONValue() } / * Creates a new JSONValue */ protected JSONValue(JSONErrorHandler errorHandler)**

Creates a new JSONValue.

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

This will read the vale from an input stream.

Returns

the last character read.

Parameters

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

reader - The reader to read from.

Throws

IOException - on IO failure.

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

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

IOException - On IO failure.

protected JSONErrorHandler getErrorHandler()

Returns

The user supplied error handler.

protected void warn(String message)

Provide a warning.

Parameters

message - The warning message.

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 IOException***This will read the value from an input stream.**Parameters**is* - The input stream to read from.*Throws**IOException* - on IO failure.**public void writeJSON(OutputStream os) throws IOException***This writes JSON to the specified OutputStream.**Parameters**os* - The outoutStream to write to.*Throws**IOException* - on IO failure.**public void writeJSON(OutputStream os, boolean compact) throws IOException***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*

IOException - on IO failure.

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 IOException

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

Throws

IOException - on IO problems.

protected char getChar(boolean handleEscapes) throws IOException

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

Parameters

handleEscapes - If true then * escape character are handled.

Throws

IOException - on IO problems.

protected void ungetChar(char c) throws IOException

Unreads the specified character so that the next call to getNextChar() will return it again.

Parameters

c - The character to unget.

protected char skipWhitespace(char c) throws IOException

Skips whitespace returning the first non whitespace character. This also sets the EOF flag.

Parameters

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

Throws

IOException

protected char skipWhitespace() throws IOException

Skips whitespace returning the first non whitespace character. This also sets the EOF flag.

Throws

IOException

protected char readUntil(String until, char c, StringBuilder sb, boolean handleEscapes) throws IOException

Reads until any of a specified set of characters occur.

*Returns**Parameters*

until - The characters to stop reading at. The stopping character will be returned unless EOF.

c - The first preread character.

sb - If not null read characters are added to this. The stopping character will not be included.

handleEscapes - True if we are reading a string that should handle escape characters.

Throws

IOException

protected char readUntil(String until, StringBuilder sb, boolean string) throws IOException

Reads until any of a specified set of characters occur.

Parameters

until - The characters to stop reading at. The stopping character will be returned unless EOF.

sb - If not null read characters are added to this. The stopping character will not be included.

string - True if we are rading a string that should be escaped.

*Throws**IOException***protected char readUntil(String until, StringBuilder sb) throws IOException***Reads until any of a specified set of characters occur.**Parameters**until* - The characters to stop reading at. The stopping character will be returned unless EOF.*sb* - If not null read characters are added to this. The stopping character will not be included.*Throws**IOException***protected boolean checkValidChar(char c, String validChars)***Returns true if c is one of the characters in validChars.**Parameters**c* - The character to check.*validChars* - The valid characters.**protected void assertChar(char a, char e, String message)***Asserts that char a equals expected char c.**Parameters**a* - The char to assert.*e* - The expected value.*message* - Failure message.**protected void assertChar(char a, String expected, String message)***Asserts that char a equals expected char c.**Parameters**a* - The char to assert.*expected* - String of valid characters.

message - Failure message.

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 IOException

Writes JSON output.

Parameters

json - The JSON output to write.

Throws

IOException - on IO failure.

protected void writeLn(String json) throws IOException

Writes JSON output plus a newline.

Parameters

json - The JSON output to write.

Throws

IOException

}

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

This wraps a Java Bean instance allowing it to be populated with data using setProperty(String, Object) methods handling all reflection calls.

public BeanInstance(Object modelInstance)

Creates a new ModelInstance.

Parameters

modellInstance - 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 JSONException

Sets a property

Parameters

property - The name of the property to set.

value - The value to set with.

Throws

JSONException - on any failure to set the property.

public Object getProperty(String property) throws JSONException

Returns the value of the specified property.

Returns

The property value.

Parameters

property - The property to return value of.

Throws

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

public Class getPropertyType(String property) throws JSONException

Returns the type of the specified property.

Returns

The class representing the property type.

Parameters

property - The property to get the type for.

Throws

JSONConversionException - if property does not exist.

}

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

Takes a JavaBean and produces a JSONObject.

public static JSONObject convertObject(Object javaBean) throws JSONConversionException

Converts a JavaBean object into a JSONObject.

Returns

A JSONObject containing all values from the JavaBean.

Parameters

javaBean - The JavaBean object to convert.

Throws

JSONConversionException - on converting failure.

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

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

JSONConversionException - on converting failure.

public static JSONValue convertValue(Object value)

Converts a value from a java value to a JSONValue.

Returns

The converted JSONValue.

Parameters

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

}

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

Creates a new JSONConversionException.

Parameters

message - The exception message

public JSONConversionException(final String message, final Throwable cause)

Creates a new JSONConversionException.

Parameters

message - The exception message

cause - The cause of this exception.

}

```
public class JSONTToJava [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 IOException, JSONConversionException

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

Returns

A populated instance of javaClass.

Parameters

jsonStream - The stream to read from.

javaClass - The java class to instantiate and populate.

Throws

IOException - on IO failures.

JSONConversionException - On JSON to Java failures.

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

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

Returns

A populated instance of javaClass.

Parameters

json - The String to read from.

javaClass - The java class to instantiate and populate.

Throws

IOException - on IO failures.

JSONConversionException - On JSON to Java failures.

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

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

Returns

A converted Java object.

Parameters

json - The json to convert to java.

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

Throws

JSONConversionException - On failure to convert.

}

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

```
}
```

6 APSToolsLib

This is a library of utilities including a service tracker that beats the (beep) out of the default one including exception rather than null response, 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.

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 requiring any other APS bundle.

6.1 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 *APSServiceUnavailableException* if the service is not available. Personally I think this is easier to work with than having to check for a null result.

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

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

Note that the third argument, which is a timeout can also be specified as an int in which case it is always in milliseconds. The string variant supports the a second word of "seconds" and "minutes" 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.

6.1.1 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 it will be the first service instance received. It will keep tracking other instances coming in, but as long as the active service does not go away it will be the one used. If the active service goes away then the the one that is at the beginning of the list of the other tracked instances will become active. If that list is empty there will be no active, which will trigger a wait for a service to become available again if `allocateService()` is called.

6.1.2 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.

6.1.3 Tracker as a wrapped service

The tracker can be used as a wrapped service:

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

This gives you a proxied *service* instance that 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 *APSNServiceAvailableException* will be thrown. This is of course a runtime exception which makes the service wrapping possible without losing the possibility to handle the case where the service is not available.

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

To get a service instance you do:

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

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

When done with the service do:

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

6.1.5 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.

6.1.5.1 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.
    }
});
```

6.1.5.2 onServiceLeaving

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

```
onServiceLeaving(new OnServiceLeaving<Service>() {
    @Override
    public void onServiceLeaving(ServiceReference service, Class serviceAPI)
```

```
throws Exception {
    // Handle the service leaving.
}
});
```

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

6.1.5.3 onActiveServiceAvailable

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

6.1.5.4 onActiveServiceLeaving

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

6.1.5.5 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
    }
});
```

6.1.5.6 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!

6.1.5.7 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.

6.2 APSLogger

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

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

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

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

If you call the `setServiceReference(serviceRef);` method on the logger then information about that service will be provided with each log.

6.3 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.

6.4 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.

7 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.

7.1 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 saveSessionIdOnResponse(HttpServletResponse response)

Saves the current session id on the specified response.

Parameters

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

}

```
public class APSLLoginHandler 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)"
```

```
public APSLLoginHandler(BundleContext context, HandlerInfo handlerInfo)
```

Creates a new VaadinLoginDialogHandler.

Parameters

context - The bundles BundleContext.

```
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.

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.

This method does not use or modify any internal state of this object! It only uses the APSUserService 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 APSUserService 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 getSessionName()*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 isValidLogin()

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 methods needs to be called when the handles is no longer needed.

}

8 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 use password authentication.

8.1 APSSimpleUserServiceAuthProvider

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.

8.2 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 need authentication use this service.

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

APS supplies an APSSimpleUserServiceAuthProvider 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.

}

9 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. Maybe that should be done via the credentials Dictionary, but what are the expected keys in there ?

9.1 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")) {
    ...
}
```

9.2 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** is however important. The service will be looking up the entry with that name! The rest of the entry 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.

9.3 APIs

```
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 APSSimpleUserServiceAuthProvider 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.

```
}
```

```
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 message.

public APSAuthMethodNotSupportedException(String message, Throwable cause)

Creates a new APSAuthMethodNotSupportedException instance.

Parameters

message - The exception message.

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 message.

```
public APSSimpleUserServiceException(String message, Throwable cause)
```

Creates a new APSSimpleUserServiceException instance.

Parameters

message - The exception message.

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 up 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.

}

10 APSDataSource

This is a service that provides named data source definitions. It does **not** provided 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".

10.1 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 up a JDBC data source.

String getName()

Returns

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

String getConnectionString()

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 **APSDDataSourceDefService** [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.

}

11 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.getConnectionDriverName());
        this.emp = this.jpaService.initialize(this.bundleContext,
"myPersistenceUnitName", props);
    }
    return this.emp;
}
...
EntityManager em = getEMP().createEntityManager();
em.getTransaction().begin();

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

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

11.1 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. The whole point with the APS project is to provide a configured platform into which you can drop applications and they adapt to their surrounding. Not unlike what JEE does, but does it milder and more flexibly being OSGi and also provides application and service specific configuration with a web gui for editing configuration. Thereby providing database connection properties from clients allows clients more flexibility in how they want to handle that. The APSDataSourceDef service can for example be used to lookup a JDBC connection definition. 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 <a href='http://docs.jboss.org/ejb3/app-

server/tutorial/extended_pc/extended.html' 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 entity manager factory. This will return null if isValid() return false!

}

12 APSJSONService

This provides exactly the same functionality 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 user of this service. The APS users 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 exists 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.

13 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 quite 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!

13.1 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 up. The more bundles and the more dependencies among them the higher the value should be. The default value is 8.

14 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 together. This is explained in detail in the APSAdminWeb documentation.

14.1 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 close.

}

15 APSDiscoveryService

This is actually a service directory that also multicasts on the network to find other instances of itself and makes the services in other instances available also.

A *service* here means anything that can be called either with an URL or a host and port. It does not specifically indicate what type of service it is. Each published service however has an id that can be used to better identify it. Basically clients of services have to know the id of the service they want and by that know what it is and how to talk to it.

15.1 APIs

```
public class APSDiscoveryPublishException extends APSRuntimeException
[se.natusoft.osgi.aps.api.net.discovery.exception] {
```

Thrown on service publish problems.

```
public APSDiscoveryPublishException(String message)
```

Creates a new APSRuntimeException instance.

Parameters

message - The exception message.

```
public APSDiscoveryPublishException(String message, Throwable cause)
```

Creates a new APSRuntimeException instance.

Parameters

message - The exception message.

cause - The cause of this exception.

```
}
```

```
public interface ServiceDescription [se.natusoft.osgi.aps.api.net.discovery.model] {
```

Describes a service.

```
String getDescription()
```

A short description of the service.

```
String getServiceId()
```

An id/name of the service.

String getVersion()*The version of the service.***String getServiceHost()***The targetHost of the service.***int getServicePort()***The targetPort of the service.***String getServiceURL()***An optional URL to the service.*

}

```
public class ServiceDescriptionProvider implements ServiceDescription
[se.natusoft.osgi.aps.api.net.discovery.model] {
```

Describes a service.

```
public ServiceDescriptionProvider() } // // Methods // /** * Returns a string representation of this object. */
public String toString()
```

*Creates a new ServiceDescription.***public String getDescription()***A short description of the service.***public void setDescription(String description)***Sets a short description of the service.**Parameters**description* - *The description to set.***public void setServiceId(String serviceId)***Sets the id of the service.**Parameters*

serviceId - The service id to set.

public void setVersion(String version)

Sets the version of the service.

Parameters

version - The version to set.

public void setServiceHost(String serviceHost)

Sets the targetHost of the service.

Parameters

serviceHost - The service targetHost to set.

public void setServicePort(int servicePort)

Sets the targetPort of the service.

Parameters

servicePort - The service targetPort to set.

Sets an url to the service.

Parameters

serviceURL - The service url to set.

}

```
public interface APSSimpleDiscoveryService [se.natusoft.osgi.aps.api.net.discovery.service] {
```

A network service discovery.

public List<ServiceDescription> getRemotelyDiscoveredServices()

Returns all remotely discovered services.

public List<ServiceDescription> getLocallyRegisteredServices()

Returns the locally registered services.

public List<ServiceDescription> getAllServices()

Returns all known services, both locally registered and remotely discovered.

public List<ServiceDescription> getService(String serviceId, String version)

Returns all discovered services with the specified id.

Parameters

serviceId - The id of the service to get.

version - The version of the service to get.

public void publishService(ServiceDescription service) throws APSDiscoveryPublishException

Publishes a local service. This will announce it to other known APSSimpleDiscoveryService instances.

Parameters

service - The description of the service to publish.

Throws

APSDiscoveryPublishException - on problems to publish (note: this is a runtime exception!).

public void unpublishService(ServiceDescription service) throws APSDiscoveryPublishException

Recalls the locally published service, announcing to other known APSSimpleDiscoveryService instances that this service is no longer available.

Parameters

service - The service to unpublish.

Throws

APSDiscoveryPublishException - on problems to publish (note: this is a runtime exception!).

}

16 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" the service is made externally available. If the *MANIFEST.MF* contains "APS-Externalizable: false" however making the service externally available is forbidden.

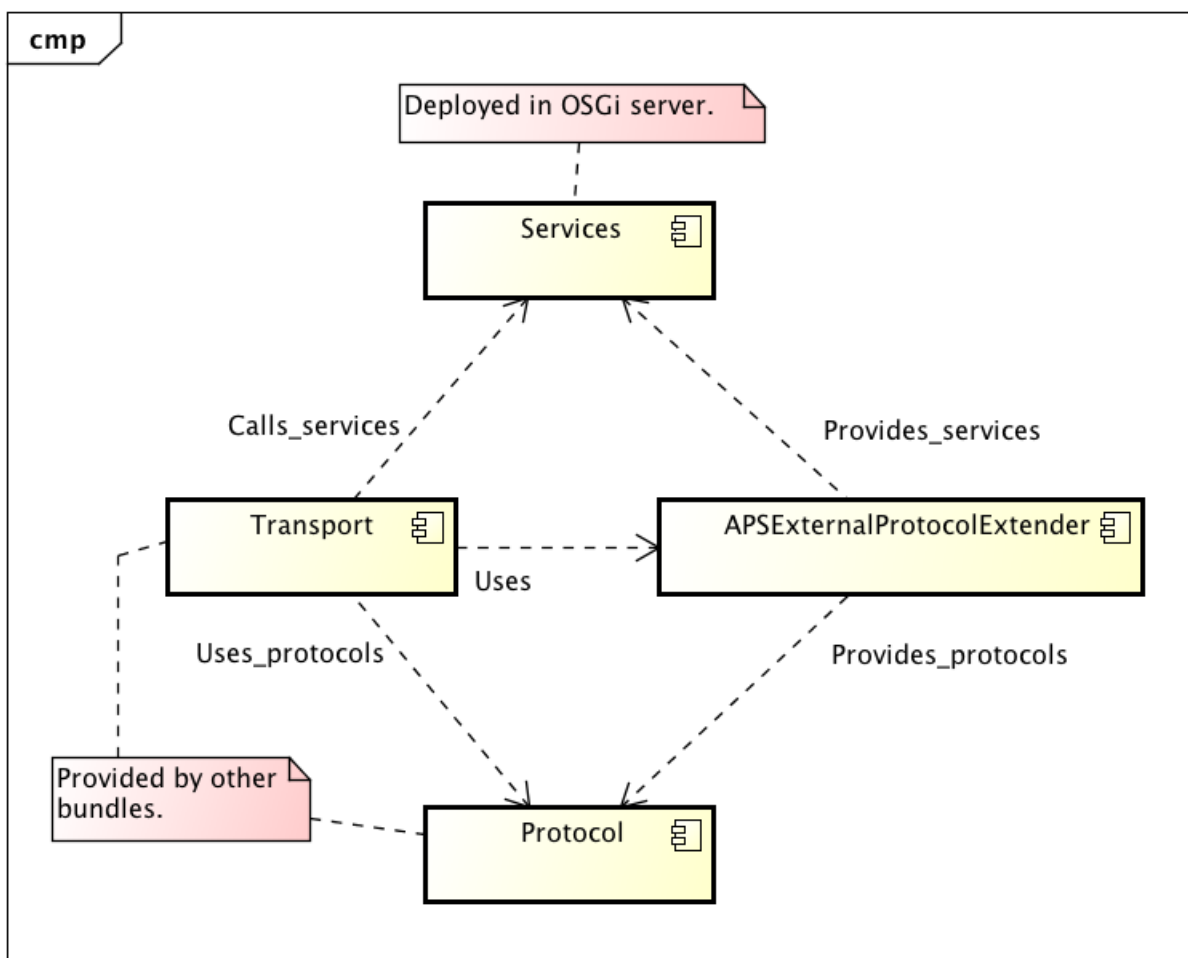
The external protocol extender also provides a configuration where services can be specified with their fully qualified name to be made externally available. If a bundle however have specifically specified false for the above manifest entry then the config entry will be ignored.

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.

16.1 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.

16.2 **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 changes for services and protocols.

16.2.1 **Protocols**

There is a base API for protocols: `RPCProtocol`. APIs for different types of protocols should extend this. There is currently only one type of protocol available: *StreamedRPCProtocol*. 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* 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.

16.2.2 **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.

16.3 **See also**

APSExtProtocolHTTPTransportProvider - Provides a HTTP transport.

APSStreamedJSONRPCProtocolProvider - Provides version 1.0 and 2.0 of JSONRPC.

16.4 **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 `APSEexternallyCallable` have gone away after it was returned, but before you do `call()` on it! In that case an `APSNServiceAvailableException` will be thrown. Note that you can register as an `APSEExternalProtocolListener` 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<APSEexternallyCallable> getCallables(String serviceName) throws RuntimeException

Returns all `APSEexternallyCallable` for the named service object.

Parameters

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

Throws

RuntimeException - If the service is not available.

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 APSEexternallyCallable getCallable(String serviceName, String serviceFunctionName)

Gets an `APSEexternallyCallable` for a specified service name and service function name.

Returns

An `APSEexternallyCallable` 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 void setArguments(Object... value)

Provides parameters to the callable using a varargs list of parameter values.

Parameters

value - A parameter value.

ReturnType call() throws Exception

Calls the service method represented by this APSExternallyCallable.

Returns

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

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 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 APSExternalProtocolExtender+APSRPCHTTPTransportProvider. 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 message.

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 thought 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>.

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.

INVALID_PARAMS

The parameters to the method are invalid.

INTERNAL_ERROR

Internal protocol error.

SERVER_ERROR

Server related errors.

REST

This means the protocol is of REST type and should return a HTTP status code.

}

public interface RPCErrors [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 int getRESTHttpStatusCode()

This should return a valid http status code if `ErrorType == REST`.

public String getMessage()

Returns an error message. 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 abstract class AbstractRPCRequest implements `RPCRequest` `[se.natusoft.osgi.aps.api.net.rpc.model]` {

This contains a parsed JSONRPC request.

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

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.

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()

REST protocol where it will be part of the URL.

Returns

The method to call. This can return null if the method is provided by other means, for example a

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.

void addParameter(Object parameter)

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

Parameters

parameter - The parameter to add.

void addNamedParameter(String name, Object parameter)

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

Parameters

name - The name of the parameter.

parameter - The parameter to add.

int getNumberOfParameters()

Returns

The number of parameters available.

<T> T getParameter(int index, Class<T> paramClass)

Returns the parameter at the specified index.

Returns

The parameter object.

Parameters

index - *The index of the parameter to get.*

paramClass - *The expected class of the parameter.*

boolean hasNamedParameters()**Returns**

true if there are named parameters available. If false the plain parameter list should be used.

Set<String> getParameterNames()**Returns**

The available parameter names.

<T> T getNamedParameter(String name, Class<T> paramClass)**Returns**

A named parameter.

Parameters

name - *The name of the parameter to get.*

paramClass - *The expected class of the parameter.*

}

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)

Factory method to create an error object.

Returns

An RPCError implementation.

Parameters

errorType - The type of the error.

message - An error message.

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

}

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 and responses written to an OutputStream.

boolean isREST()

Returns true if the protocol is a REST protocol.

List<RPCRequest> parseRequests(String serviceQName, InputStream requestStream) 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.

requestStream - The stream to parse request from.

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.

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

Writes an error response.

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.

RPCError createRESError(int httpStatusCode)

Returns an RPCError for a REST protocol with a http status code.

Parameters

httpStatusCode - The http status code to return.

RPCError createRESError(int httpStatusCode, String message)

Returns an RPCError for a REST protocol with a http status code.

Parameters

httpStatusCode - The http status code to return.

message - An error message.

}

17 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.StreamedRPCProtocolService` and registering it as an OSGi service. The `StreamedRPCProtocolService` 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.

17.1 Example

Here is an example calling the `APSPlatformService` with JSONRPC 2.0 using curl:

```
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"
}
```

The APSPlatformService is a plain OSGi service that provides data with JavaBean setters and getters. This simple example only works if you have disabled the "requireAuthentication" configuration (network/rpc-http-transport).

17.2 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.

17.3 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.

17.4 See Also

Also look at the documentation for *APSExternalProtocolExtender*.

18 APSGroups

Provides network groups where named groups can be joined as members and then send and receive data messages to the group. This is based on multicast and provides a verified multicast delivery with acknowledgements of receive to the sender and resends if needed. The sender will get an exception if not all members receive all data. Member actuality is handled by members announcing themselves relatively often and will be removed when an announcement does not come in expected time. So if a member dies unexpectedly (network goes down, etc) its membership will resolve rather quickly. Members also tries to inform the group when they are doing a controlled exit.

Please note that this does not support streaming! That would require a far more complex protocol. APSGroups waits in all packets of a message before delivering the message.

18.1 OSGi service usage

The APSGroupsService can be used as an OSGi service and as a standalone library. This section describes the service.

18.1.1 Getting the service

```
APSServiceTracker<APSGroupsService> apsGroupsServiceTracker =
    new APSServiceTracker<APSGroupsService>(bundleContext, APSConfigService.class,
        APSServiceTracker.LARGE_TIMEOUT);
APSGroupsService apsGroupsService = apsGroupsServiceTracker.getWrappedService();
```

18.1.2 Joining a group

```
GroupMember groupMember = apsGroupsService.joinGroup("mygroup");
```

18.1.3 Sending a message

To send a message you create a message, get its output stream and write whatever you want to send on that output stream, close it and then send it. *Note* that since the content of the message is any data you want, all members of the groups must know how the data sent looks like. In other words, you have to define your own message protocol for your messages. Note that you can wrap the OutputStream in an ObjectOutputStream and serialize any java object you want.

```
Message message = groupMember.createNewMessage();
OutputStream msgDataStream = message.getOutputStream();
try {
    ...
    msgDataStream.close();
    groupMember.sendMessage(message);
}
catch (IOException ioe) {
    ...
}
```

Note that the `groupMember.sendMessage(message)` does throw an `IOException` on failure to deliver the message to all members.

18.1.4 Receiving a message

To receive a message you have to register a message listener with the GroupMember object.

```
MessageListener msgListener = new MyMsgListener();
groupMember.addMessageListener(myMsgListener);
```

and then handle received messages:

```
public class MyMsgListener implements MessageListener {
    public void messageReceived(Message message) {
        InputStream msgDataStream = message.getInputStream();
        ...
    }
}
```

18.1.5 Leaving a group

```
apsGroupsService.leaveGroup(groupMember);
```

18.2 Library usage

The bundle jar file can also be used as a library outside of an OSGi server, with an API that has no other dependencies than what is in the jar. The API is then slightly different, and resides under the `se.natusoft.apsgroups` package.

18.2.1 Setting up

```
APSGroups apsgroups = new APSGroups(config, logger);
apsGroups.connect();
```

The config passed as argument to APSGroups will be explained further down under "Configuration".

The *logger* is an instance of an implementation of the APSGroupsLogger interface. Either you provide your own implementation of that or you use the APSGroupsSystemOutLogger implementation.

18.2.2 Joining a group

```
GroupMember groupMember = apsgroups.joinGroup("mygroup");
```

18.2.3 Sending and receiving messages

Sending and receiving works exactly like the OSGi examples above.

18.2.4 Leaving a group

```
apsGroups.leaveGroup(groupMember);
```

18.2.5 Shutting down

```
apsGroups.disconnect();
```

18.3 Net time

All APSGroups instances connected will try to sync their time. I call this synced time "net time".

It works like this: When an APSGroups instance comes up it waits a while for NET_TIME packets. If it gets such a packet then it enters receive mode and takes the time in the received NET_TIME packet and stores a diff to that time and local time. This diff can then be used to translate back and forth between local and net time. If no such packet arrives in expected time it enters send mode and starts sending NET_TIME packets itself using its current net time. If a NET_TIME packet is received when in send mode it directly goes over to listen mode. If in listen mode and no NET_TIME packet comes in reasonable time it goes over to send mode. So among all instances on the network only one is responsible for sending NET_TIME. If that instance leaves then there might be a short fight for succession, but

it will resolve itself rather quickly.

The GroupMember contains a few *create** methods to produce a *NetTime* object instance. See the API further down for more information on these.

18.4 Configuration

18.4.1 OSGi service

The OSGi service provides a configuration model that gets managed by the APSConfigService. It can be configured in the APS adminweb (<http://host:port/apsadminweb/>). Here is a screenshot of the config admin:

The screenshot shows the 'Application Platform Services Admin Web' interface. The 'Configuration' tab is selected. The left sidebar shows a tree view with 'Config Environments' and 'Configurations'. Under 'Configurations', 'aps' is expanded, showing 'persistence', 'network', and 'service'. 'service' is further expanded, showing 'rpc-http-transport', 'discovery', and 'groups'. 'groups' is selected. The main content area shows the configuration for 'Config ID: se.natusoft.osgi.aps.groups'. It includes a dropdown for 'Edit for configuration environment:' set to 'default'. Below this are five configuration fields: 'multicastaddress' (224.0.0.1), 'multicastport' (58100), 'sendtimeout' (120), 'resendinterval' (5), and 'memberannounceinterval' (10). Each field has a description. At the bottom are 'Save' and 'Cancel' buttons.

Application Platform Services Admin Web [Refresh](#)

About Configuration User Admin Remote Services

Config ID: se.natusoft.osgi.aps.groups

Edit for configuration environment:
default

multicastaddress
The multicast address to use.
224.0.0.1

multicastport
The multicast target port to use.
58100

sendtimeout
The number of seconds to allow for a send of a message before timeout.
120

resendinterval
The number of seconds to wait before a packet is resent if not acknowledged. `sendTimeout / resendInterval` = the number or resends before giving up.
5

memberannounceinterval
The interval in seconds that members announce that they are (sill) members. If a member has not announced itself again within this time other members of the group will drop the member.
10

Save Cancel

18.4.2 Library

The library wants an implementation of the APSGroupsConfig interface as its first argument to APSGroups(config, logger) constructor. Either you implement your own or use the APSGroupsConfigProvider implementation. This is a plain java bean with both setters and getters for the config values. It comes with quite reasonable default values. It

contains exactly the same properties as shown in the picture above.

18.5 APIs

public interface **APSGroupsService** [se.natusoft.osgi.aps.api.net.groups.service] {

A service that lets clients send data reliable to all members of a group on any host. There is no limit on the size of the data sent, but that said I wouldn't send MB:s of data!

GroupMember joinGroup(String name) throws IOException

Joins a group.

Returns

A GroupMember that provides the API for sending and receiving data in the group.

Parameters

name - The name of the group to join.

Throws

java.io.IOException - The unavoidable one!

void leaveGroup(GroupMember groupMember) throws IOException

Leaves as member of group.

Parameters

groupMember - The GroupMember returned when joined.

Throws

java.io.IOException - The unavoidable one!

}

public interface **GroupMember** [se.natusoft.osgi.aps.api.net.groups.service] {

This is the API for APSGroupsService members received when they join a group. It is used to send and receive data messages to/from the group.

void addMessageListener(MessageListener listener)

Adds a listener for incoming messages.

Parameters

listener - The listener to add.

void removeMessageListener(MessageListener listener)

Removes a listener for incoming messages.

Parameters

listener - The listener to remove.

Message createNewMessage()

Creates a new Message to send. Use the sendMessage() method when ready to send it.

void sendMessage(Message message) throws IOException

Sends a previously created message to all current members of the group. If this returns without an exception then all members have received the message.

Parameters

message - The message to send.

Throws

java.io.IOException - On failure to reach all members.

UUID getMemberId()

Returns

The ID of the member.

List<String> getMemberInfo()

Returns information about members.

NetTime getNow()

Returns

The current time as net time.

NetTime createFromNetTime(long netTimeMillis)

Creates from milliseconds in net time.

Parameters

netTimeMillis - The net time milliseconds to create a *NetTime* for.

NetTime createFromNetTime(Date netTimeDate)

Creates from a Date in net time.

Parameters

netTimeDate - The Date in net time to create a *NetTime* for.

NetTime createFromLocalTime(long localTimeMillis)

Creates from milliseconds in local time.

Parameters

localTimeMillis - The local time milliseconds to create a *NetTime* for.

NetTime createFromLocalTime(Date localTimeDate)

Creates from a Date in local time.

Parameters

localTimeDate - The Date in local time to create a *NetTime* for.

}

public interface **Message** [se.natusoft.osgi.aps.api.net.groups.service] {

*This represents a complete message containing any data you want to send to the group. You provide the message with data using the *OutputStream*, and read message data using the *InputStream*.*

OutputStream getOutputStream()

*Returns an *OutputStream* to write message on. Multiple calls to this will return the same *OutputStream*!*

InputStream getInputStream()

*Returns an *InputStream* for reading the message. Multiple calls to this will return new *InputStream*:s starting from the beginning!*

UUID getId()

Returns the id of this message.

String getMemberId()

Returns

id of member as a string.

String getGroupName()**Returns**

The name of the group this message belongs to.

}

public interface **MessageListener** [se.natusoft.osgi.aps.api.net.groups.service] {

For listening on messages from the group.

public void messageReceived(Message message)

Notification of received message.

Parameters

message - The received message.

}

public interface **NetTime** extends Serializable [se.natusoft.osgi.aps.api.net.groups.service] {

This represents a common network time between members for handling date and time data. The net time is synchronized between all members. Each receiver of net time diffs it with local time and stores the diff so that they can convert to/from local/net time.

public long getNetTime()

Returns the number of milliseconds since Januray 1, 1970 in net time.

public Date getNetTimeDate()

Returns the net time as a Date.

public Calendar getNetTimeCalendar()

Returns the net time as a Calendar.

public Calendar getNetTimeCalendar(Locale locale)

Returns the net time as a Calendar.

Parameters

locale - The locale to use.

public Date getLocalTimeDate()

Converts the net time to local time and returns as a Date.

public Calendar getLocalTimeCalendar()

Converts the net time to local time and returns as a Calendar.

public Calendar getLocalTimeCalendar(Locale locale)

Converts the net time to local time and returns as a Calendar.

Parameters

locale - The locale to use.

}

19 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 1.0 version of the JSONRPC protocol are described at <http://json-rpc.org/wiki/specification>.

The 2.0 version of the JSONRPC protocol are described at <http://jsonrpc.org/spec.html>.

19.1 See also

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

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