APS Simple User Service Provider

User Guide

Version: 1.0.0

Author: Tommy Svensson

Table of Contents

1 APSSimpleUserService

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

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

1.1 Basic example

To login a user do something like this:

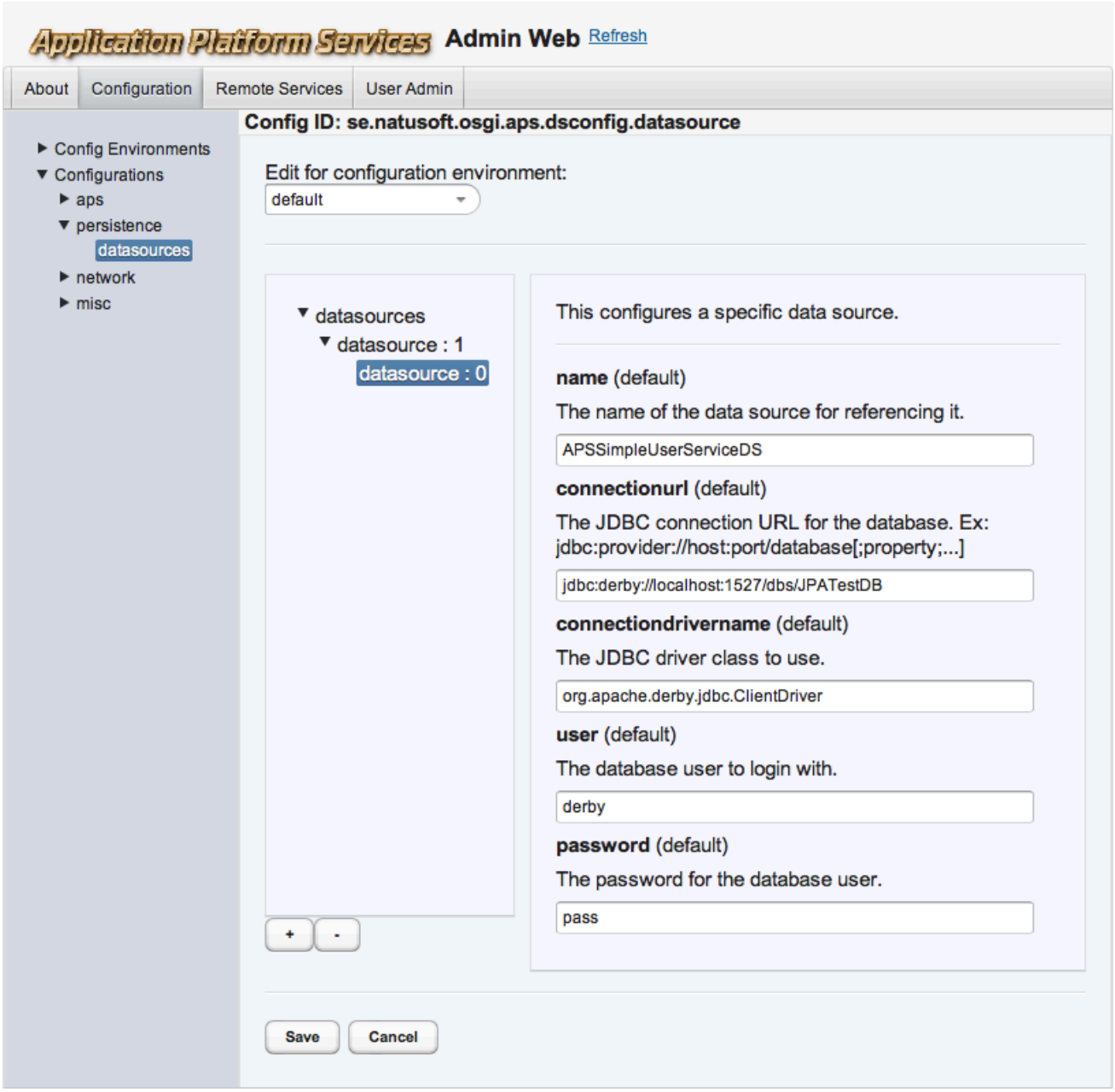
```
    APSSimpleUserService userService ...
    ...
    User user = userService.getUser(userId);
    if (user == null) {
        throw new AuthException("Bad login!");
    }
    if (!userService.authenticateUser(user, password,
APSSimpleUserService.AUTH_METHOD_PASSWORD)) {
        throw new AuthException("Bad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

1.2 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* in this example should be configured in the *persistence/dsRefs* config where you provide a name and a datasource reference. The service will be looking up the entry with that name, and use the specified datasource! For example:

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

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

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

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

1.3 Troubleshooting

If you have managed to lock yourself out of /apsadminweb as described above then I suggest editing the *APSFilesystemService root*/filesystems/se.natusoft.osgi.aps.core.config.service.APSConfigServiceProvider/apsconfig-se.natusoft.aps.adminweb-1.0.properties file and changeing the following line:

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

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

1.4 JDBC Drivers

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

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

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

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

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

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

or

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

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

1.5 APIs

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

This is the API of a simple user service that provide basic user handling that will probably be enough in many cases, but not all.

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

public static final String AUTH_METHOD_PASSWORD = "password"

Password authentication method for authenticateUser().

public Role getRole(String roleId)

Gets a role by its id.

Returns

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

Parameters

roleId *- The id of the role to get.*

public User getUser(String userId)

Gets a user by its id.

Returns

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

Parameters

userId *- The id of the user to get.*

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

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

Returns

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

Parameters

user *- The User object representing the user to authenticate.*

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

authMethod *- Specifies what authentication method is wanted.*

}

---

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

Admin API for APSSimpleUserService.

public RoleAdmin createRole(String name, String description)

Creates a new role.

Returns

a new Role object representing the role.

Parameters

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

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

public void updateRole(Role role)

Updates a role.

Parameters

role *- The role to update.*

public void deleteRole(Role role)

Deletes a role.

Parameters

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

public List<RoleAdmin> getRoles()

Returns all available roles.

public UserAdmin createUser(String id)

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

Returns

A User object representing the new user.

Parameters

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

public void updateUser(User user)

Updates a user.

Parameters

user - *The user to update.*

public void deleteUser(User user)

Deletes a user.

Parameters

user - *The user to delete.*

public List<UserAdmin> getUsers()

Returns all users.

public void setUserAuthentication(User user, String authentication)

Sets authentication for the user.

Parameters

user - *The user to set authentication for.*

authentication - *The authentication to set.*

}

---

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

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

public APSAuthMethodNotSupportedException(String message)

Creates a new APSAuthMethodNotSupportedException instance.

Parameters

message - *The exception messaging.*

public APSAuthMethodNotSupportedException(String message, Throwable cause)

Creates a new APSAuthMethodNotSupportedException instance.

Parameters

message - *The exception messaging.*

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

}

---

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

Indicates a problem with the APSSimpleUserService.

public APSSimpleUserServiceException(String message)

Creates a new APSSimpleUserServiceException instance.

Parameters

message - *The exception messaging.*

public APSSimpleUserServiceException(String message, Throwable cause)

Creates a new APSSimpleUserServiceException instance.

Parameters

message - *The exception messaging.*

cause - *The cause of the exception.*

}

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

This defines a role.

public String getId()

Returns

The id of the role.

public String getDescription()

Returns

A description of the role.

public boolean hasRole(String roleName)

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

Parameters

roleName - *The name of the role to check for.*

boolean isMasterRole()

Returns

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

}

---

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

Provides update API for Role.

public void setDescription(String description)

Changes the description of the role.

Parameters

description - *The new description.*

public List<Role> getRoles()

Returns all sub roles for this role.

public void addRole(Role role)

Adds a sub role to this role.

Parameters

role *- The role to add.*

public void removeRole(Role role)

Removes a sub role from this role.

Parameters

role *- The role to remove.*

public void setMasterRole(boolean masterRole)

Sets whether this is a master role or not.

Parameters

masterRole *- true for master role.*

}

---

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

This defines a User.

public String getId()

Returns the unique id of the user.

public boolean isAuthenticated()

Returns true if this user is authenticated.

public boolean hasRole(String roleName)

Returns true if the user has the specified role name.

Parameters

roleName *- The name of the role to check for.*

public Properties getUserProperties()

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

Please note that the returned properties are read only!

public static final String USER_NAME = "name"

Optional suggestion for user properties key.

public static final String USER_PHONE = "phone"

Optional suggestion for user properties key.

public static final String USER_PHONE_WORK = "phone.work"

Optional suggestion for user properties key.

public static final String USER_PHONE_HOME = "phone.home"

Optional suggestion for user properties key.

public static final String USER_EMAIL = "email"

Optional suggestion for user properties key.

}

---

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

Provides update API for the User.

public List<Role> getRoles()

Returns all roles for this user.

public void addRole(Role role)

Adds a role to this user.

Parameters

role *- The role to add.*

public void removeRole(Role role)

Removes a role from this user.

Parameters

role *- The role to remove.*

public void addUserProperty(String key, String value)

Adds a user property.

Parameters

key *- The key of the property.*

value *- The value of the property.*

public void removeUserProperty(String key)

Removes a user property.

Parameters

key - *The key of the property to remove.*

public void setUserProperties(Properties properties)

Sets properties for the user.

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

Parameters

properties - *The properties to set.*

}
