# PicketLink IDM

# Reference Guide

## 1.3.0.GA

by Boles#aw Dawidowicz and Jeff Yu

## What this Book Covers

This book aims to help you become familiar with PicketLink IDM component

Part I 'Design and Architecture' introduces the the main concepts behind framework design

Part II 'Concepts behind API to SPI mappings' describes framework concepts in a more detailed way and explains relationship between API and SPI layer.

Part III 'Quick Start' provides a reader with best way to start playing with the framework

Part IV 'Configuration' describes framework configuration.

Part VI 'IdentityStore Implementations' provides detailed view on provided IdentityStore implementations.

Part VII 'IdentityStoreRepository Implementations' provides detailed view on provided IdentityStoreRepository implementations.

Part VIII 'Attributes' describes how attributes can be used in the API.

Part IX 'Credentials' describes how credentials can be used in the API.

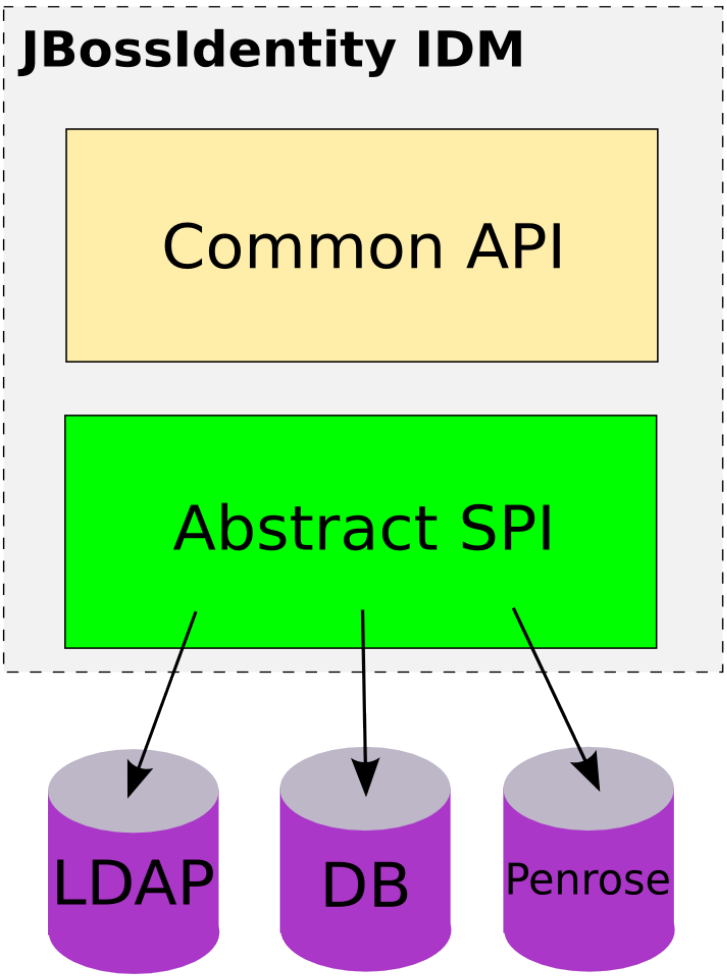
Part X 'Deployment' provides additional insight on how IDM component can be leveraged inside JEE container.

# Part I. Design and Architecture

# Main Concepts

PicketLink IDM aims to provide a common identity model for various JBoss projects. As every project has it's own specific needs it's hard to design a common API and identity model that will fit all of them. Therefore PicketLink IDM architecture consists of two main parts:

- **Core SPI** with an abstract identity model that provides the flexibility for defining different identity object types and possible relationships between them.
- **Common API** with a simpler identity model that fits most common use cases. Identity model has more strictly defined object types and possible relationships.




**PicketLink IDM architecture**

Too much abstraction in the API layer would confuse people using the framework. One of the goals is to make the design easily extendible. Framework adopters should be able to remove the API layer and reuse core SPI implementation if needed.

# API Model

The API operates on the identity model that is defined by a set of following interfaces:

- **org.picketlink.idm.api.IdentityType** - is a parent interface for Group and Identity
- **org.picketlink.idm.api.Identity** - represents Identity which can be a user (within organization) or a machine (in authentication or security use case)
- **org.picketlink.idm.api.Group** - represents typed Group
- **org.picketlink.idm.api.GroupType** - represents type of a Group. It can be an organization, organization unit, administration group, global role, community or any other entity.
- **org.picketlink.idm.api.Role** - represents one to one relationship between Identity and Group. Role has a type. The idea behind the concept is described below.
- **org.picketlink.idm.api.RoleType** - represents type of a Role.
- **org.picketlink.idm.api.Attribute** - represents attribute connected with IdentityType (Group or Identity). Can have many complex type values (text or binary). AttributeDescription describes such properties of Attribute like: name, type of values, readonly, multivalued, required.
- **org.picketlink.idm.api.Credential** - represents credential connected with Identity.
- **org.picketlink.idm.api.CredentialType** - represents type of a Credential. For example it can be text password or binary certificate.

# Groups and Roles

Groups are entities that can contain other group or identity objects. They can be associated in a tree like organizational structures. Those don't need to be hierarchical only as single group can be a member of many other groups (can have many parents). Possible relationships between groups are shaped with group types. It can be configured which different group types can be associated or even which group types can or can not contain identity objects. Groups have unique names per group type. This means you can have two groups with the same name but different group type.

Roles are direct typed connections between Identity and Group objects. If you think about a sentence: "**John** is the **Manager** of **XX Team"** what matters is the context. So "John (Identity) is the Manager (RoleType) of XX Team (Group)". The whole sentence describes the Role that John has. This type of information is hard to map with typical Group object as John can be a manager of several different groups and other identities (Marry, Jack, Stan...) can have the same RoleType in context of different groups (XY Team, YY Team). Within each Realm (concept of Realms is described later) we can define several RoleType objects with unique names. Each Role defines a unique combination of Identity, Group and RoleType within Realm. Role concept is very powerful but its not natural in all identity store types. While quite easy to map in a relational database it doesn't fit into every LDAP tree present in organizations. Because of this Role support is optional in the API level

# SPI (Abstract) Model

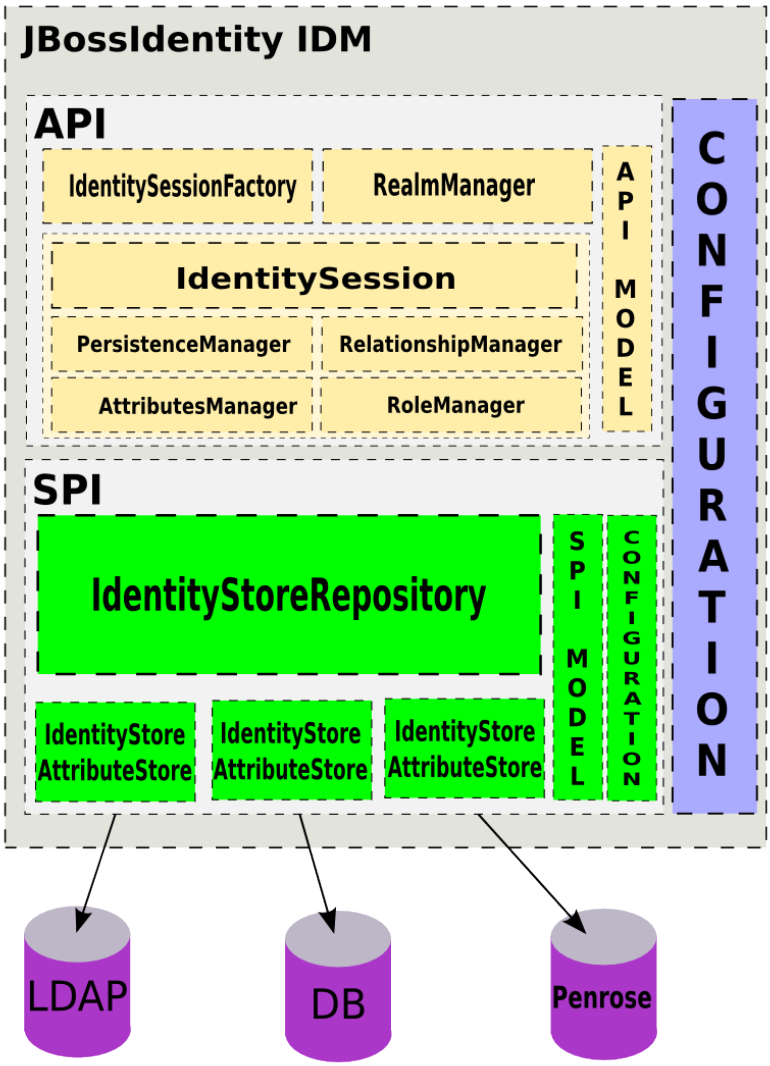
SPI Model contains following interfaces:

- **org.picketlink.idm.spi.model.IdentityObject** - represents identity object. Contains information about object name and type (IdentityObjectType). Name and IdentityObjectType pair should be unique within realm.
- **org.picketlink.idm.spi.model.IdentityObjectType** - represents identity object type. Name of IdentityType is unique.
- **org.picketlink.idm.spi.model.IdentityObjectAttribute** - attribute assigned to IdentityObject
- **org.picketlink.idm.spi.model.IdentityObjectCredential** - credential assigned to IdentityObject
- **org.picketlink.idm.spi.model.IdentityObjectCredentialType** - represents type of IdentityObjectCredential.
- **org.picketlink.idm.spi.model.IdentityObjectRelationship** - Directional relationship between two IdentityObject objects. Relationship is directional as it keeps information about from and to IdentityObject. Each IdentityObjectRelationship has a type (IdentityObjectRelationshipType) and can have a name (not required).
- **org.picketlink.idm.spi.model.IdentityObjectRelationshipType** - named type of relationship

Model described above is very flexible as IdentityObjectType is able to map any kind of entities. Identity object and Group/GroupType objects are only one of many possible options (API is a subset of SPI possibilities). IdentityObjectRelationship defines a connection between any two IdentityObject objects. Each IdentityObjectRelationship has a type. To map previously described API two IdentityObjectRelationshipType objects are needed. One to map normal MEMBERSHIP like between an Identity and a Group or Group and Group objects. Second one to map Role concept. For API Role - RoleType refers to the name of the IdentityObjectRelationship. In default Hibernate implementation possible names of IdentityObjectRelationship are kept in a separate table. All of those can be easily redefined to support different kind of API.

# Architecture

The most important part of architecture is a split between the API and the SPI.




**PicketLink IDM architecture**

API part contains of following interfaces:

- **Realm** - described later in this document. Groups configuration of several identity stores and exposes all of them with one consistent identity model.
- **IdentitySessionFactory** - Main entry point in the API. Enables to create/get IdentitySession for a given Realm
- **IdentitySession** - Session that groups all identity management operation. Contains transaction support and exposes four managers that handle all identity management operations.

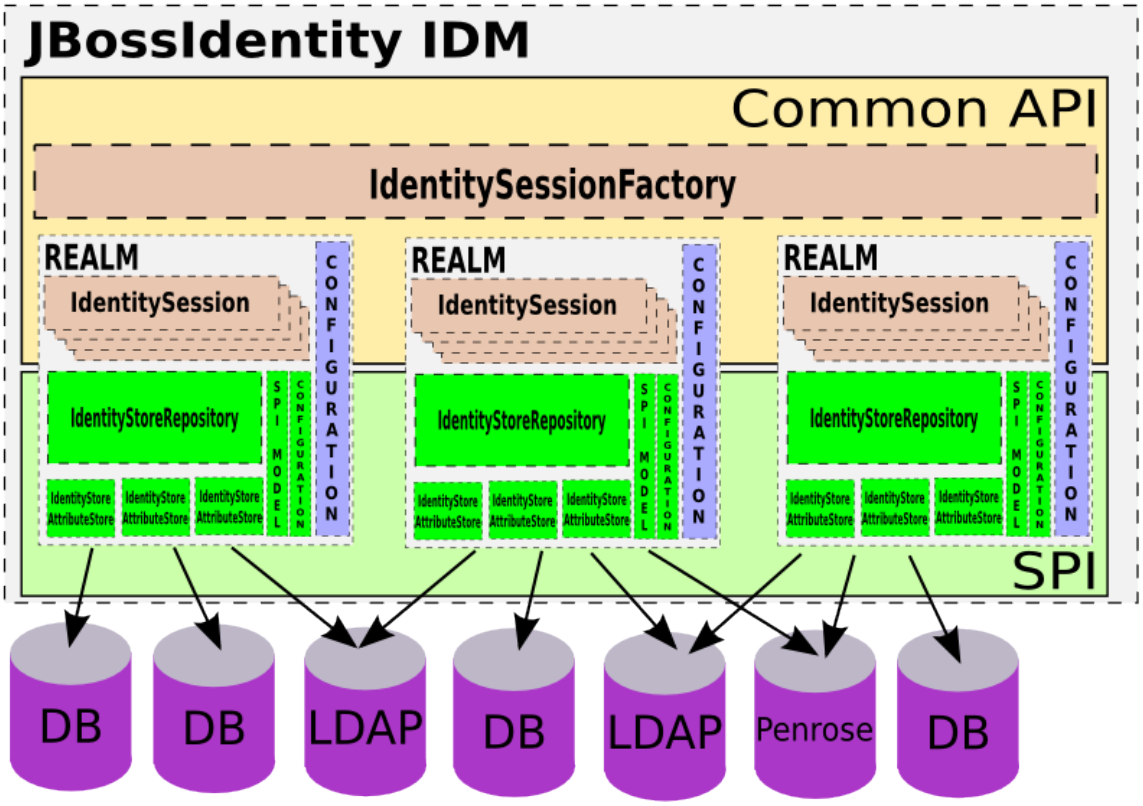
- **PersistenceManager** - Operates witin IdentitySession. Performs all operations on Identity and Group objects. Create/Remove/Find
- **RelationshipManager** - Operates witin IdentitySession. Associate and deassociate Identity and Group objects. Find Idenity and Group objects depending on their relationships
- **RoleManager** - Operates witin IdentitySession. Operations on Role objects. Optional feature.
- **AttributeManager** - Operates witin IdentitySession. Manages Identity and Group (IdentityType objects) attributes. Each configured attribute is described with AttributeDescription interface
- **SearchCriteria** - Enables to apply additional conditions to search operations. May be leveraged to receive results sorted, paginated or filtered with attributes.

SPI part contains of following interfaces

- **IdentityStoreSession** - Session that groups all identity management operations within identity persistence stores.
- **IdentityStoreSessionFactory** - Entry point in the SPI to initialize IdentityStoreSession inside IdentityStore or AttributeStore.
- **IdentityStoreInvocationContext** - IdentityStoreSession aware context object that is passed during any invocation of AttributeStore or IdentityStore methods. Thanks to this actual store implementation doesn't need to be aware of current session state. Therefore one instance of AttributeStore or IdentityStore can be invoked by different realms at the same time.
- **AttributeStore** - Exposes operations on identity store with attributes. This is a separate interface as in multi store configuration scenario, profile may need to be stored outside of actual data store for a given identity (LDAP + DB)
- **IdentityStore** - Extends AttributeStore. Implementation of this interface performs operations on the real data store.
- **IdentityStoreRepository** - Extends IdentityStore. Groups several IdentityStore objects and exposes operations on them within single interface for the API. The implementation is responsible for aggregate identity objects from many configured underlaying IdentityStore objects and map different IdentityObjectTypes between them. The place where the whole magic happens
- **IdentityObjectSearchControl** - Enables to apply additional conditions to search operations. May be leveraged to receive results sorted, paginated or filtered withattributes.

# Realms

The purpose of a Realm is to group configuration of several identity stores. IdentitySession exposes operations within single Realm




**PicketLink IDM architecture**

# Part II. Concepts behind API to SPI mappings

# Introduction

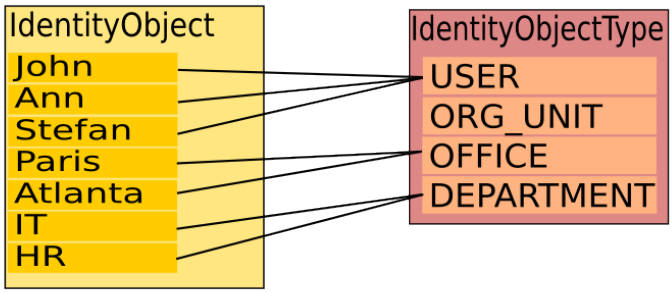
The most confusing part of the framework is probably connection between API and SPI. This part will try to explain how operations on API model are translated into the SPI. It will also enable to dive in the API and framework capabilities by looking on realcode examples

# User and Group

Objects represented by User and Group interfaces are managed by PersistenceManager. In the example below 3 users and 4 groups are created.

```
PersistenceManager pm = identitySession.getPersistenceManager();
User johnUser = pm.createUser("John");
User annUser = pm.createUser("Ann");
User stefanUser = pm.createUser("Stefan");

String OFFICE = "OFFICE";
String DEPARTMENT = "DEPARTMENT";

Group parisOffice = pm.createGroup("Paris", OFFICE);
Group atlantaOffice = pm.createGroup("Atlanta", OFFICE);

assertEquals(OFFICE, parisOffice.getGroupType());

Group itDep = pm.createGroup("IT", DEPARTMENT);
Group hrDep = pm.createGroup("HR", DEPARTMENT);
```

At the SPI level both Group and User are mapped as IdentityObject. What differentiate them is the IdentityObjectType. IdentityObjectType "USER" is mapped to represent User objects while other are mapped to represent different Group types names.


IdentityObject
John
Ann
Stefan
Paris
Atlanta
IT
HR
IdentityObjectType
USER
ORG_UNIT
OFFICE
DEPARTMENT

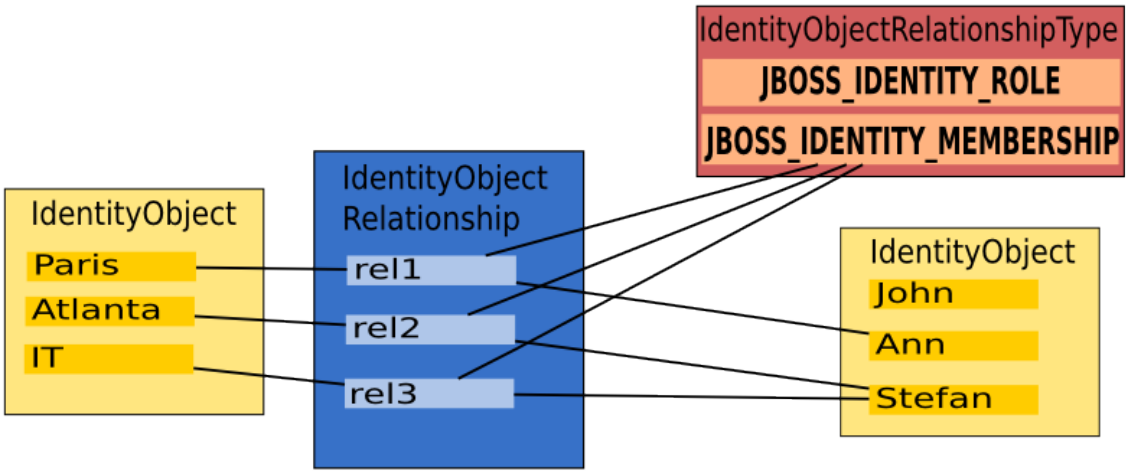
# Associations

Group and Users can be associated. This represents simple relationship that can be described like "user John belongs to Group IT". Association can be created between Group and User or between two Groups.

Those operations are managed by RelationshipManager:

```
RelationshipManager rm = identitySession.getRelationshipManager();

rm.associateUser(parisOffice, annUser);
rm.associateUser(atlantaOffice, stefanUser);
rm.associateUser(itDep, stefanUser);

assertTrue(rm.isAssociated(parisOffice, annUser));
```

At the SPI level this is mapped to IdentityObjectRelationship entity. This relationship has a type. Simple assotiations can be marked with a type named "JBOSS_IDENTITY_MEMBERSHIP" (this is implementation detail) which describes simple membership. Note that IdentityObjectRelationship creates a connection between any two IdentityObject entities.



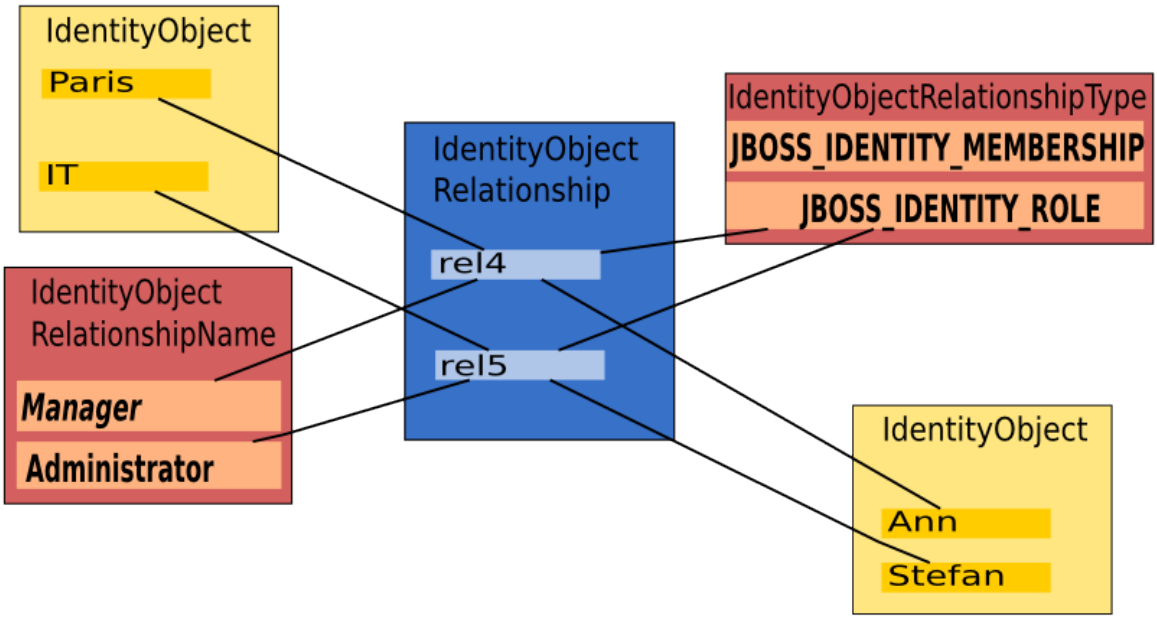
# Role

Roles are direct typed connections between Identity and Group objects. If you think about a sentence: "John is the Manager of XX Team" what matters is the context. So "John (Identity) is the Manager (RoleType) of XX Team (Group)". The whole sentence describes the Role that John has. This type of information is hard to map with typical Group object as John can be a manager of several different groups and other identities (Marry, Jack, Stan...) can have the same RoleType in context of different groups (XY Team, YY Team). Within each Realm (concept of Realms is described later) we can define several RoleType objects with unique names. Each Role defines a unique combination of Identity, Group and RoleType within Realm.

Roles are managed with RoleManager interface:

```
RoleManager roleManager = identitySession.getRoleManager();

roleManager.createRoleType("manager");
RoleType adminRT = roleManager.createRoleType("administrator");

Role role1 = roleManager.createRole("manager", annUser.getId(), parisOffice.getId());
roleManager.createRole(adminRT, stefanUser, itDep);

assertTrue(roleManager.hasRole(stefanUser, itDep, adminRT));
```

At the SPI level the main difference between plain association is that IdentityObjectRelationship has a IdentityObjectRelationshipName which is simple mapping of a RoleType used in the API




What is important to note about the Role concept is that it is not natural in all kinds of identity stores. Entities represented on attached figures are easy to map in the database. However in store like LDAP typical relationships are represanted in a more plain manner. For example:

```
dn: uid=admin,ou=People,o=test,dc=portal,dc=example,dc=com
objectclass: top
objectclass: inetOrgPerson
objectclass: person
uid: admin
cn: Java Duke
sn: Duke
userPassword: admin
mail: email@email.com

dn: cn=Administrators,ou=Groups,o=test,dc=portal,dc=example,dc=com
objectClass: top
objectClass: groupOfNames
cn: Administrators
description: Portal admin role
member: uid=admin,ou=People,o=test,dc=portal,dc=example,dc=com
```

The whole relationship between User "admin" and Group "Administrators" is described by one attribute value ("member"). In such typical LDAP tree shape there is no place to store additional information that are needed to describe Role shown above. Obviously it is possible to shape LDAP tree in a way that will allow such a mapping but in most cases it is not possible to redesign already used LDAP server tree.

# Part III. Quick Start

# Test Cases

One of the best ways to get familiar with the PicketLink IDM component is to look at the source code. You will find link to the subversion repository in the project webpage. There are couple of quite meaningful testcases there. One of the best to start with is OrganizationTest under 'idm-testsuite' module . It contains two example identity structures. One mapping hierarchical organization of Red Hat and JBoss projects and the other describes theoretical portal tree for ACME company.

# Examples

As PicketLink IDM is a Maven2 based project it is very easy to leverage it from this build system. There is a ready to use Maven2 example project in the svn. It contains three sample test cases for with following configurations:

- database setup
- LDAP setup
- mixed LDAP + database setup

Sample project uses embedded OpenDS and HSQLDB so there is no need for any additional setup to be able to play with the API.

# Needed files

Although the best way to start playing with the framework is to look at Maven2 sample project mentioned above lets list minimal set of configuration files. To setup the basic framework core depending on hibernate IdentityStore two files will be needed

**idm-config.xml** - that will set proper configuration for all framework components described in section above. Sample one below.

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-identity xmlns="urn:picketlink:idm:config:v1_0_0_ga"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="urn:picketlink:idm:config:v1_0_0_ga identity-config.xsd">
   <realms>
      <realm>
         <id>realm::JBossIdentityExample_SampleRealm</id>
         <repository-id-ref>Sample Repository</repository-id-ref>
         <identity-type-mappings>
            <user-mapping>USER</user-mapping>
         </identity-type-mappings>
      </realm>
   </realms>
   <repositories>
      <repository>
         <id>Sample Repository</id>
         <class>org.picketlink.idm.impl.repository.WrapperIdentityStoreRepository</class>
         <external-config/>
         <default-identity-store-id>Sample DB Store</default-identity-store-id>
         <default-attribute-store-id>Sample DB Store</default-attribute-store-id>
      </repository>
   </repositories>
   <stores>
      <attribute-stores/>
      <identity-stores>
         <identity-store>
            <id>Sample DB Store</id>
            <class>org.picketlink.idm.impl.store.hibernate.HibernateIdentityStoreImpl</class>
            <external-config/>
            <supported-relationship-types>
               <relationship-type>JBOSS_IDENTITY_MEMBERSHIP</relationship-type>
               <relationship-type>JBOSS_IDENTITY_ROLE</relationship-type>
```

```
        </supported-relationship-types>
        <supported-identity-object-types>
          <identity-object-type>
            <name>USER</name>
            <relationships>
              <relationship>
                <relationship-type-ref>JBOSS_IDENTITY_ROLE</relationship-type-ref>
                <identity-object-type-ref>GROUP</identity-object-type-ref>
              </relationship>
            </relationships>
            <credentials>
              <credential-type>PASSWORD</credential-type>
            </credentials>
            <attributes>
              <attribute>
                <name>picture</name>
                <mapping>user.picture</mapping>
                <type>binary</type>
                <isRequired>false</isRequired>
                <isMultivalued>false</isMultivalued>
                <isReadOnly>false</isReadOnly>
              </attribute>
            </attributes>
            <options/>
          </identity-object-type>
          <identity-object-type>
            <name>ORGANIZATION</name>
            <relationships>
              <relationship>
                <relationship-type-ref>JBOSS_IDENTITY_ROLE</relationship-type-ref>
                <identity-object-type-ref>USER</identity-object-type-ref>
              </relationship>
              <relationship>
              <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
                <identity-object-type-ref>USER</identity-object-type-ref>
              </relationship>
              <relationship>
              <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
                <identity-object-type-ref>GROUP</identity-object-type-ref>
              </relationship>
            </relationships>
            <credentials/>
            <attributes/>
            <options/>
```

```
        </identity-object-type>
        <identity-object-type>
            <name>GROUP</name>
            <relationships>
                <relationship>
                    <relationship-type-ref>JBOSS_IDENTITY_ROLE</relationship-type-ref>
                    <identity-object-type-ref>USER</identity-object-type-ref>
                </relationship>
                <relationship>
            <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
                    <identity-object-type-ref>USER</identity-object-type-ref>
                </relationship>
                <relationship>
            <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
                    <identity-object-type-ref>GROUP</identity-object-type-ref>
                </relationship>
            </relationships>
            <credentials/>
            <attributes/>
            <options/>
        </identity-object-type>
    </supported-identity-object-types>
    <options>
        <option>
            <name>hibernateConfiguration</name>
            <value>hibernate-jboss-identity.cfg.xml</value>
        </option>
        <option>
            <name>populateRelationshipTypes</name>
            <value>true</value>
        </option>
        <option>
            <name>populateIdentityObjectTypes</name>
            <value>true</value>
        </option>
        <option>
            <name>allowNotDefinedAttributes</name>
            <value>true</value>
        </option>
        <option>
            <name>isRealmAware</name>
            <value>true</value>
        </option>
    </options>
```

```
            </identity-store>
        </identity-stores>
    </stores>
</jboss-identity>
```

**hibernate.cfg.xml** - hibernate SessionFactory setup

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>


        <property name="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</
property>

        <property name="show_sql">false</property>
        <property name="cache.use_second_level_cache">true</property>
        <property name="cache.use_query_cache">true</property>


        <property name="current_session_context_class">thread</property>

        <!--<property name="connection.datasource"></property>-->

        <property name="hibernate.connection.url">jdbc:hsqldb:mem:unit-testing-jpa1</property>
        <property name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="hibernate.dialect">org.hibernate.dialect.HSQLDialect</property>
        <property name="hibernate.hbm2ddl.auto">create-drop</property>
        <property name="hibernate.connection.username">sa</property>
        <property name="hibernate.connection.password"></property>

    </session-factory>
</hibernate-configuration>
```

# Part IV. Configuration

# IdentitySessionFactory

IdentitySessionFactory interface is a main entry point into the API. Default implementation IdentitySessionFactoryImpl has two constructors:

- public IdentitySessionFactoryImpl(IdentityConfigurationMetaData configMD) throws Exception
- public IdentitySessionFactoryImpl(File configFile) throws Exception

Framework configuration can be defined in two ways. It can be passed as implementation of a set of metadata interfaces grouped in org.picketlink.idm.spi.configuration.metadata package. Main one is IdentityConfigurationMetaData.

Other possibility is to use xml configuration file that will be unmarshaled into JAXB model (org.picketlink.idm.impl.configuration.jaxb2.generated package) and used to create IdentityConfigurationMetaData object. XML configuration is described by identity-config.xsd file. It is good to take a look at the example organization-test-config.xml that is used in the testsuite.

# XML Configuration

```
<jboss-identity xmlns="urn:jboss:identity:idm:config:v1_0_alpha"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:jboss:identity:idm:config:v1_0_alpha identity-config.xsd">
    <realms/>
    <repositories/>
    <stores/>
</jboss-identity>
```

Identity XML configuration can be divided into three parts:

- **<stores/>** - defines IdentityStore and AttributeStore instances
- **<repositories/>** - defines IdentityStoreRepository instances
- **<realms/>** - defines identity realms

# IdentityStore

This part is represented by **<stores><identity-store>...** element

```
<stores>
  <attribute-stores/>
  <identity-stores>
    <identity-store> ... </identity-store>
    <identity-store> ... </identity-store>
  </identity-stores>
 </stores>
```

**Note**

**<stores/>** element contains **<identity-store>** and **<attribute-store>** elements. Currently **<attribute-store>** configuration is ignored so only **<identity-store>** elements can be configured.

```
<identity-store>
  <id>Sample Hibernate Store</id>
  <class>org.picketlink.idm.impl.store.hibernate.HibernateIdentityStoreImpl</class>
  <external-config/>
  <supported-relationship-types>
    <relationship-type>
      JBOSS_IDENTITY_MEMBERSHIP
    </relationship-type>
    <relationship-type>
      JBOSS_IDENTITY_ROLE
    </relationship-type>
  </supported-relationship-types>
  <supported-identity-object-types>
    <identity-object-type>
    ...
    </identity-object-type>
```

```
    ...


    <identity-object-type>
    ...
    </identity-object-type>
  </supported-identity-object-types>
  <options>
    <option>
      <name>persistenceUnit</name>
      <value>jboss-identity-model-xxx</value>
    </option>
    <option>
      <name>otherOption</name>
      <value>value1</value>
      <value>value2</value>
      <value>value3</value>
    </option>
  </options>
</identity-store>
```

**<identity-store>** element:

- **id** - IdentityStore id
- **class** - IdentityStore class name
- **external-config** - external configuration file used by IdentityStore
- **supported-relationship-types** - IdentityObjectRelationshipType names that are supported by this IdentityStore. JBOSS_IDENTITY_MEMBERSHIP is standard value used by default framework implementation for membership type relationships (between two Group objects) and JBOSS_IDENTITY_ROLE is standard value for Role type memberships (named relationships)
- **supported-identity-object-types** - configuration of IdentityObjectType objects mapped by IdentityStore
- **options** - other IdentityStore configuration options

```
<supported-identity-object-types>
  <identity-object-type>
```

```
    <name>OFFICE</name>
    <relationships>
      <relationship>
        <relationship-type-ref>
          JBOSS_IDENTITY_MEMBERSHIP
        </relationship-type-ref>
        <identity-object-type-ref>
          IDENTITY
        </identity-object-type-ref>
      </relationship>
      <relationship>
        <relationship-type-ref>
          JBOSS_IDENTITY_MEMBERSHIP
        </relationship-type-ref>
        <identity-object-type-ref>
          CONFERENCE_ROOM
        </identity-object-type-ref>
      </relationship>
    </relationships>
    <credentials/>
    <attributes/>
    <options/>
  </identity-object-type>

  <identity-object-type>
    <name>IDENTITY</name>
    <relationships>
      <relationship>
        <relationship-type-ref>
          JBOSS_IDENTITY_ROLE
        </relationship-type-ref>
        <identity-object-type-ref>
          COMMUNITY
        </identity-object-type-ref>
      </relationship>
    </relationships>
    <credentials>
      <credential-type>PASSWORD</credential-type>
      <credential-type>BINARY</credential-type>
    </credentials>
    <attributes>
      <attribute>
        <name>picture</name>
        <mapping>user.picture</mapping>
```

```
          <type>binary</type>
          <isRequired>false</isRequired>
          <isMultivalued>false</isMultivalued>
          <isReadOnly>false</isReadOnly>
        </attribute>
      </attributes>
    </identity-object-type>
</supported-identity-object-types>
```

<identity-object-type> element:

- **name**- IdentityObjectType name
- **relationships** - relationships in which IdentityObjectType can be parent side. **<relationship-type-ref>** must point to one of values from **<supported-relationship-types>** . **<identity-object-type-ref>** must be one of **<identity-object-type><name>** values configured in this IdentityStore.
- **credentials** - IdentityObjectCredentialType names allowed for this IdentityObjectType
- **attributes** - allowed attribute mappings. Each contains:
  - **name** - attribute name
  - **mapping** - real name to be used inside IdentityStore. For example LDAP attribute name
  - **type** - either "binary" or "text" value
  - **isRequired** - if attribute cannot have no values
  - **isMultivalued** - if attribute can have many values
  - **isReadonly** - if attribute values can be modified
- **options** - other options for IdentityObjectType configuration

# IdentityStoreRepository

This section is represented by **<repositories><repository>** element

```
<repositories>

  <repository>
    <id>X</id>
    <class>
      org.picketlink.idm.impl.repository.WrapperIdentityStoreRepository
    </class>
    <external-config/>
    <default-identity-store-id>
      Hibernate Identity Store
    </default-identity-store-id>
    <default-attribute-store-id>
      Hibernate Identity Store
    </default-attribute-store-id>
    <options>
  </repository>

  <repository>
    <id>Y</id>
    <class>
      org.picketlink.idm.impl.repository.FallbackIdentityStoreRepository
    </class>
    <external-config/>
    <default-identity-store-id>
      Hibernate Identity Store
    </default-identity-store-id>
    <default-attribute-store-id>
      Hibernate Identity Store
    </default-attribute-store-id>
    <identity-store-mappings>
      <identity-store-mapping>
        <identity-store-id>
          Hibernate Identity Store
        </identity-store-id>
        <identity-object-types>
          <identity-object-type>
            PROJECT
```

```
            </identity-object-type>
            <identity-object-type>
              PEOPLE
            </identity-object-type>
          </identity-object-types>
          <options/>
        </identity-store-mapping>
        <identity-store-mapping>
          <identity-store-id>
            LDAP Identity Store
          </identity-store-id>
          <identity-object-types>
            <identity-object-type>
              IDENTITY
            </identity-object-type>
            <identity-object-type>
              ORGANIZATION
            </identity-object-type>
          </identity-object-types>
          <options/>
        </identity-store-mapping>
      </identity-store-mappings>
      <options/>
    </repository>

</repositories>
```

**<repository>** element contains:

- **id** - IdentityStoreRepository id.
- **class** - class name of IdentityStoreRepository implementation.
- **external-config** - external configuration file used by IdentityStoreRepository.
- **default-identity-store-id** - id of configured IdentityStore to be used by default.
- **default-attribute-store-id** - id of configured AttributeStore (or IdentityStore) to be used by default
- **identity-store-mappings** - optional element. Mappings between IdentityObjectType names and IdentityStore ids.

# Realm

This section is represented by **<realms><realm>** element

```
<realm>
  <id>realm::RedHat_DB</id>
  <repository-id-ref>RedHat Repository DB</repository-id-ref>
  <identity-type-mappings>
     <identity-mapping>IDENTITY</identity-mapping>
  </identity-type-mappings>
</realm>
```

**<realm>** element contains:

- **id** - realm id
- **repository-id-ref** - id of configured IdentityStoreRepository
- **identity-type-mappings**
  - **identity-mapping** - name of IdentityObjectType that should be maped as Identity object on the API side

# Realm Templates

Realm can be marked as a template. This means that if a different realm name is requested using the API and such name contains template realm as a prefix then this realm configuration will be used

```
<realm>
  <id>idm_realm</id>
  <repository-id-ref>DefaultRepository</repository-id-ref>
  <identity-type-mappings>
    <user-mapping>USER</user-mapping>
  </identity-type-mappings>
  <options>
    <option>
      <name>template</name>
      <value>true</value>
    </option>
  </options>
</realm>
```

In this example for "idm_realm_foo" framework will return "idm_realm" configuration. Request for "foo" realm won't return valid configuration.

```
<realms>
  <realm>
    <id>idm_realm</id>
    <repository-id-ref>DefaultRepository</repository-id-ref>
    <identity-type-mappings>
      <user-mapping>USER</user-mapping>
    </identity-type-mappings>

  </realm>
</realms>
<repositories>...</repositories>
<stores>...</stores>
```

```
  <options>
    <option>
      <name>defaultTemplate</name>
      <value>idm_realm</value>
    </option>
  </options>
```

The "defaultTemplate" option defines idm_realm as the one that should be used when requested realm name is not found in configuration

# API Cache

At the realm level IDM API cache layer can be enabled

```
<realm>
  <id>idm_realm</id>
  <repository-id-ref>DefaultRepository</repository-id-ref>
  <identity-type-mappings>
    <user-mapping>USER</user-mapping>
  </identity-type-mappings>
  <options>
    <option>
      <name>cache.providerClass</name>
      <value>JBossCacheAPICacheProviderImpl</value>
    </option>
  </options>
</realm>
```

Cache related options :

- **cache.providerClass** - class implementing APICacheProvider interface
- **cache.providerRegistryName** - id of instantiated APICacheProvider implementation stored in configuration registry
- **cache.scope** - namespace used to store entities in cache. Can be "realm" (default) or "session".

# Part V. IdentityStore Implementations

# Overview

This part describes different IdentityStore implementations that comes with the framework and their configuration options

# Hibernate IdentityStore

## 22.1. Class Name

org.picketlink.idm.impl.store.hibernate.HibernateIdentityStoreImpl

## 22.2. Overview

HibernateIdentityStoreImpl maps PicketLink IDM SPI model into Hibernate entities. This enables to use any RDBMS supported by Hibernate as identity persistence store (IdentityStore). Because of flexibility that ORM gives this IdentityStore implementation support all of the optional design concepts like role management. It can be used as the default IdentityStore together with other more limited implementations. For example in combination with LDAP IdentityStore it can handle IdentityObject attributes that are not supported in LDAP schema. In such configuration part of IdentityObject profile will be stored in LDAP and part in relational database. To learn more about such setup please read FallbackIdentityStoreRepository documentation. In current version implementation doesn't have any caching mechanism besides of what can be set in hibernate configuration

## 22.3. Configuration Options

- **hibernateConfiguration** - the hibernate configuration file that will be used to create SessionFactory
- **hibernateSessionFactoryJNDIName** - JNDI name of hibernate SessionFactory that will be used to obtain it
- **hibernateSessionFactoryRegistryName** - name of hibernate SessionFactory placed in the IdentityConfigurationRegistry that will be used to obtain it
- **addHibernateMappings** - of set to true all annotated hibernate model classes will be added to the hibernate configuration before SessionFactory is created
- **populateRelationshipTypes** - true/false - Populate configured <supported-relationship-types> (IdentityObjectRelationshipType in SPI model) during IdentityStore initialization. Default value is 'false'
- **populateIdentityObjectTypes** - true/false - Populate configured <supported-identity-object-types> (IdentityObjectType in SPI model) during IdentityStore initialization. Default value is 'false'
- **allowNotDefinedAttributes** - true/false - Allow to set IdentityObject attributes that are not specified in <identity-object-type> configuration. Such attributes are assumed to have "text" type and many values. Default value is 'false'.
- **isRealmAware** - true/false - If set to true HibernateIdentityStoreImpl will create separate namespaces for different Realms from which method invocations come. This means that

each IdentityObject, IdentityObjectRelationship and IdentityObjectRelationshipName will be connected and only accessible with a realm name in which it was created. Entities representing IdentityObjectType, IdentityObjectCredentialType and IdentityObjectRelationshipType are always same for all realms and not affected with this option. Default value is 'false'.

- **manageTransactionDuringBootstrap** - true/false - indicate that transactions should be managed manually during store bootstrap when initial entities are created.
- **allowNotDefinedIdentityObjectTypes** - true/false - indicate that store won't check for a given IdentityObjectType configuration and if one is not specified it will be just lazily created. This option enables to have minimal xml config without definition of all constraints in relationship between types.

## 22.4. Sample Configuration

```
<identity-store>
  <id>Hibernate Identity Store</id>
  <class>org.picketlink.idm.impl.store.hibernate.HibernateIdentityStoreImpl</class>
  <external-config/>
  <supported-relationship-types>
    <relationship-type>JBOSS_IDENTITY_MEMBERSHIP</relationship-type>
    <relationship-type>JBOSS_IDENTITY_ROLE</relationship-type>
  </supported-relationship-types>
  <supported-identity-object-types>
    <identity-object-type>
      <name>IDENTITY</name>
      <relationships/>
      <credentials>
        <credential-type>PASSWORD</credential-type>
      </credentials>
        <attributes>
          <attribute>
            <name>user.name.given</name>
            <mapping>user.name.given</mapping>
            <type>text</type>
            <isRequired>false</isRequired>
            <isMultivalued>false</isMultivalued>
            <isReadOnly>false</isReadOnly>
          </attribute>
          <attribute>
            <name>picture</name>
            <mapping>user.picture</mapping>
```

```
        <type>binary</type>
        <isRequired>false</isRequired>
        <isMultivalued>false</isMultivalued>
        <isReadOnly>false</isReadOnly>
       </attribute>
     </attributes>
     <options/>
   </identity-object-type>
   <identity-object-type>
     <name>ORGANIZATION</name>
     <relationships>
       <relationship>
         <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
         <identity-object-type-ref>IDENTITY</identity-object-type-ref>
       </relationship>
       <relationship>
         <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
         <identity-object-type-ref>ORGANIZATION</identity-object-type-ref>
       </relationship>
       <relationship>
         <relationship-type-ref>JBOSS_IDENTITY_ROLE</relationship-type-ref>
         <identity-object-type-ref>IDENTITY</identity-object-type-ref>
       </relationship>
     </relationships>
     <credentials/>
     <attributes/>
     <options/>
   </identity-object-type>
 </supported-identity-object-types>
 <options>
   <option>
     <name>hibernateConfiguration</name>
     <value>hibernate-jboss-identity.cfg.xml</value>
   </option>
   <option>
     <name>populateRelationshipTypes</name>
     <value>true</value>
   </option>
   <option>
     <name>populateIdentityObjectTypes</name>
     <value>true</value>
   </option>
   <option>
     <name>allowNotDefinedAttributes</name>
```

```
      <value>true</value>
    </option>
    <option>
      <name>isRealmAware</name>
      <value>true</value>
    </option>
    <option>
      <name>allowNotDefinedAttributes</name>
      <value>true</value>
    </option>
  </options>
</identity-store>
```

In case 'addHibernateMappings' option is not set to true hibernate configuration need to list all annotated model classes:

```
<mapping resource="mappings/HibernateRealm.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectCredentialBinaryValue.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectAttributeBinaryValue.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObject.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectCredential.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectCredentialType.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectAttribute.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectType.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectRelationship.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectRelationshipType.hbm.xml"/>
<mapping resource="mappings/HibernateIdentityObjectRelationshipName.hbm.xml"/>
```

# LDAP IdentityStore

## 23.1. Class Name

org.picketlink.idm.impl.store.ldap.LDAPIdentityStoreImpl

## 23.2. Overview

LDAPIdentityStoreImpl provides support for LDAP as identity persistence store (IdentityStore). At this stage the implementation is a bit limitted:

- Role management (IdentityObjectRelationshipName) is not supported
- Only "text" attribute type can be mapped
- Only "PASSWORD" <credential-type> can be mapped

## 23.3. Configuration

<identity-object-type><options>

- **idAttributeName** - attribute name under which IdentityObject name is specified. Required.
- **passwordAttributeName** - attribute name under which IdentityObject password is specified. Optional.
- **ctxDNs** - DN that will be used as context for IdentityObject searches. More than one value can be specified.
- **allowCreateEntry** - true/false - Specify if new IdentityObject can be created.
- **createEntryAttributeValues** - defines a set of ldap attributes that will be set on IdentityObject entry creation. Values are in "name=value" format. This enables to fulfill LDAP schema requirements.Default is false
- **parentMembershipAttributeName** - LDAP attribute that defines children of IdentityObject. This will be used to retrieved relationships from IdentityObject entry. Option is required if IdentityObjectType can be part of relationship.
- **isParentMembershipAttributeDN** - defines if values of attribute defined in parentMembershipAttributeName are fully qualified LDAP DNs.
- **allowEmptyMemberships** - defines if IdentityObject entry can have no members. Sometimes it is not allowed by LDAP schema.
- **parentMembershipAttributePlaceholder** - if LDAP schema doesn't allow empty memberships this value will be used as a placeholder. IdentityObject specified here won't be recognized as a member and ignored

- **childMembershipAttributeName** - LDAP attribute that defines parents of IdentityObject. This will be used to retrieved relationships from IdentityObject entry. Good example of such attribute in LDAP schema is 'memberOf'
- **childMembershipAttributeDN** - defines if values of attribute defined in childMembershipAttributeName are fully qualified LDAP DNs.
- **childMembershipAttributeVirtual** - specifies if attribute defined in 'childMembershipAttributeName' is a real attribute that can be updated or virtual one which value is managed by a directory and should not be updated
- **entrySearchFilter** - ldap filter to search IdentityObject with. {0} will be substitute with IdentityObject name. Example filter can look like this: "(uid={0})". This substitution behavior comes from the standard DirContext.search(Name, String, Object, SearchControls cons) method
- **entrySearchScope** - defines a search scope. Values can be "subtree" and "object"
- **enclosePasswordWith** - if specified password will be surunted with a given chars before update
- **passwordEncoding** - if specified password will be encoded before update. For example Microsoft Active Directory requires password to be enclosed with "" and encoded using 'UTF-16LE' for update.
- **passwordUpdateAttributeValues** - list of attributes that should be changed during password update

**<identity-store><options>**

- **providerURL** - LDAP connection URL. For example "ldap://localhost:389"
- **adminDN** - LDAP entry used to connect to the server.
- **adminPassword** - password related to adminDN
- **searchTimeLimit** -searchTimeLimit for LDAP search operations in miliseconds. Default value is 10000.
- **customJNDIConnectionParameters** - list of additional 'key=value' parameters that will be used to create JNDI context. Can be usefull to use additional JNDI options.
- **customSystemProperties** - list of 'key=value' properties that will be added using System.setProperty() method. This can be used to configure LDAP JNDI connection pooling which is set per JVM
- **externalJNDIContext** - name that will be used to perform JDNI lookup to grab JNDI connection context
- **sortExtensionSupported** - if set to "false" will disable the use of sort extension that triggers sort on LDAP server side

- **pagedResultsExtensionSupported** - if set to "true" will enable paged search. Searches that exceed size defined by "pagedResultsExtensionSize" option will be splitted into few smaller ones.
- **pagedResultsExtensionSize** - defines page size for "pagedResultsExtensionSupported" option.
- **createMissingContexts** - if set to "true" LDAP store will try to create missing LDAP contexts defined in the configuration.

## 23.4. Sample Configuration

```
<identity-store>
  <id>Sample LDAP Store</id>
  <class>org.picketlink.idm.impl.store.ldap.LDAPIdentityStoreImpl</class>
  <external-config/>
  <supported-relationship-types>
    <relationship-type>JBOSS_IDENTITY_MEMBERSHIP</relationship-type>
  </supported-relationship-types>
  <supported-identity-object-types>
    <identity-object-type>
      <name>IDENTITY</name>
      <relationships/>
      <credentials>
        <credential-type>PASSWORD</credential-type>
      </credentials>
      <attributes>
        <attribute>
          <name>phone</name>
          <mapping>telephoneNumber</mapping>
          <type>text</type>
          <isRequired>false</isRequired>
          <isMultivalued>false</isMultivalued>
          <isReadOnly>false</isReadOnly>
        </attribute>
          <attribute>
          <name>description</name>
          <mapping>description</mapping>
          <type>text</type>
          <isRequired>false</isRequired>
          <isMultivalued>false</isMultivalued>
          <isReadOnly>false</isReadOnly>
```

```
      </attribute>
      <attribute>
        <name>carLicense</name>
        <mapping>carLicense</mapping>
        <type>text</type>
        <isRequired>false</isRequired>
        <isMultivalued>false</isMultivalued>
        <isReadOnly>false</isReadOnly>
      </attribute>
    </attributes>
    <options>
      <option>
        <name>idAttributeName</name>
        <value>uid</value>
      </option>
      <option>
        <name>passwordAttributeName</name>
        <value>password</value>
      </option>
      <option>
        <name>ctxDNs</name>
        <value>ou=People,o=test,dc=example,dc=com</value>
      </option>
      <option>
        <name>allowCreateEntry</name>
        <value>true</value>
      </option>
      <option>
        <name>createEntryAttributeValues</name>
        <value>objectClass=top</value>
        <value>objectClass=inetOrgPerson</value>
        <value>sn= </value>
        <value>cn= </value>
      </option>
    </options>
  </identity-object-type>
  <identity-object-type>
    <name>ORGANIZATION</name>
    <relationships>
      <relationship>
        <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
        <identity-object-type-ref>IDENTITY</identity-object-type-ref>
      </relationship>
      <relationship>
```

```
        <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
        <identity-object-type-ref>ORGANIZATION</identity-object-type-ref>
      </relationship>
      <relationship>
        <relationship-type-ref>JBOSS_IDENTITY_MEMBERSHIP</relationship-type-ref>
        <identity-object-type-ref>GROUP</identity-object-type-ref>
      </relationship>
    </relationships>
    <credentials/>
    <attributes/>
    <options>
      <option>
        <name>idAttributeName</name>
        <value>cn</value>
      </option>
      <option>
        <name>ctxDNs</name>
        <value>ou=Organizations,o=test,dc=example,dc=com</value>
      </option>
      <option>
        <name>allowCreateEntry</name>
        <value>true</value>
      </option>
      <option>
        <name>membershipAttributeName</name>
        <value>member</value>
      </option>
      <option>
        <name>isMembershipAttributeDN</name>
        <value>true</value>
      </option>
      <option>
        <name>allowEmptyMemberships</name>
        <value>true</value>
      </option>
      <option>
        <name>createEntryAttributeValues</name>
        <value>objectClass=top</value>
        <value>objectClass=groupOfNames</value>
      </option>
    </options>
  </identity-object-type>
</supported-identity-object-types>
<options>
```

```
    <option>
      <name>providerURL</name>
      <value>ldap://localhost:10389</value>
    </option>
    <option>
      <name>adminDN</name>
      <value>cn=Directory Manager</value>
    </option>
    <option>
      <name>adminPassword</name>
      <value>password</value>
    </option>
    <option>
      <name>searchTimeLimit</name>
      <value>10000</value>
    </option>
  </options>
</identity-store>
```

# Minimal Configuration

The main role of configuration is to define relationship between separate framework components. It also enables to specify a lot of meta data information describing possible connections between IdentityObject types. It is however possible to not define all those meta data information and let the framework to be maximum permissive about allowed operations and lazily create not defined types:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-identity xmlns="urn:picketlink:idm:config:v1_0_0_ga"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="urn:picketlink:idm:config:v1_0_0_ga identity-config.xsd">
  <realms>
    <realm>
      <id>realm::FlexibleRealm</id>
      <repository-id-ref>Flexible Repo</repository-id-ref>
      <identity-type-mappings>
        <user-mapping>USER</user-mapping>
      </identity-type-mappings>
    </realm>
  </realms>
  <repositories>
    <repository>
      <id>Flexible Repo</id>
      <class>org.picketlink.idm.impl.repository.WrapperIdentityStoreRepository</class>
      <external-config/>
      <default-identity-store-id>Hibernate Identity Store</default-identity-store-id>
      <default-attribute-store-id>Hibernate Identity Store</default-attribute-store-id>
      <options>
        <option>
          <name>allowNotDefinedAttributes</name>
          <value>true</value>
        </option>
        <option>
          <name>allowNotDefinedIdentityObjectTypes</name>
          <value>true</value>
        </option>
      </options>
    </repository>
  </repositories>
  <stores>
```

```
    <attribute-stores/>
    <identity-stores>
      <identity-store>
        <id>Hibernate Identity Store</id>
        <class>org.picketlink.idm.impl.store.hibernate.HibernateIdentityStoreImpl</class>
        <external-config/>
        <supported-relationship-types>
          <relationship-type>JBOSS_IDENTITY_MEMBERSHIP</relationship-type>
          <relationship-type>JBOSS_IDENTITY_ROLE</relationship-type>
        </supported-relationship-types>
        <supported-identity-object-types/>
        <options>
          <option>
            <name>hibernateSessionFactoryJNDIName</name>
            <value>java:/jbossidentity/HibernateStoreSessionFactory</value>
          </option>
          <option>
            <name>populateRelationshipTypes</name>
            <value>true</value>
          </option>
          <option>
            <name>populateIdentityObjectTypes</name>
            <value>true</value>
          </option>
          <option>
            <name>isRealmAware</name>
            <value>true</value>
          </option>
          <option>
            <name>allowNotDefinedAttributes</name>
            <value>true</value>
          </option>
          <option>
            <name>allowNotDefinedIdentityObjectTypes</name>
            <value>true</value>
          </option>
        </options>
      </identity-store>
    </identity-stores>
  </stores>
</jboss-identity>
```

# Part VI. IdentityStoreRepository Implementations

# WrapperIdentityStoreRepository

## 25.1. ClassName

org.picketlink.idm.impl.repository.WrapperIdentityStoreRepository

## 25.2. Behaviour

Simply wrapps single AttributeStore and IdentityStore and pass all method invocations

## 25.3. Sample Configuration

```
<repository>
  <id>Sample Repository</id
  <class>org.picketlink.idm.impl.repository.WrapperIdentityStoreRepository</class>
  <external-config/>
  <default-identity-store-id>LDAP Store</default-identity-store-id>
  <default-attribute-store-id>LDAP Store</default-attribute-store-id>
</repository>
```

# FallbackIdentityStoreRepository

## 26.1. ClassName

org.picketlink.idm.impl.repository.FallbackIdentityStoreRepository

## 26.2. Behaviour

Resolves proper IdentityStore from IdentityObjectType mapping and delegates method invocation. For relationship related methods, if both IdentityObjectTypes are not mapped in one store, repository will try to sync them and associate in defaultIdentityStore. For attributes that are not mapped inside mapped IdentityStore repository will try to assign those attributes in defaultAttributeStore.

## 26.3. Configuration Options

- **allowNotDefinedAttributes** - if mapped IdentityStore doesn't support any attribute that was passed in method invocation FallbackIdentityStoreRepository will try to store it in defaultAttributeStore. If this option is set to true such attribute will be passed to defaultAttributeStore even if it is not mapped there.

## 26.4. Sample Configuration

```
<repository>
  <id>Repository XYZ</id>
  <class>org.picketlink.idm.impl.repository.FallbackIdentityStoreRepository</class>
  <external-config/>
  <default-identity-store-id>Identity Store XX</default-identity-store-id>
  <default-attribute-store-id>Identity Store XX</default-attribute-store-id>
  <identity-store-mappings>
    <identity-store-mapping>
      <identity-store-id>Identity Store XX</identity-store-id>
      <identity-object-types>
        <identity-object-type>DIVISION</identity-object-type>
        <identity-object-type>PROJECT</identity-object-type>
        <identity-object-type>PEOPLE</identity-object-type>
      </identity-object-types>
      <options/>
    </identity-store-mapping>
    <identity-store-mapping>
```

```
        <identity-store-id>Identity Store YY</identity-store-id>
        <identity-object-types>
          <identity-object-type>IDENTITY</identity-object-type>
          <identity-object-type>ORGANIZATION</identity-object-type>
          <identity-object-type>ORGANIZATION_UNIT</identity-object-type>
          <identity-object-type>DEPARTMENT</identity-object-type>
        </identity-object-types>
        <options>
          <option>
            <name>readOnly</name>
            <value>true</value>
          </option>
         </options>
      </identity-store-mapping>
    </identity-store-mappings>
    <options>
      <option>
        <name>allowNotDefinedAttributes</name>
        <value>true</value>
      </option>
    </options>
</repository>
```

The **readOnly** options specifies that the only write operation performed on the store will be password update.

# Part VII. Attributes

# API

On the API level each IdentityType object (Identity and Group) can have associated Attribute objects. All operations are exposed by AttributesManager interface. Each attribute is described with AttributeDescription that contains its properties such as:

- **name** - attribute name
- **readonly** - if attribute values can be changed
- **multivalued** - if attribute can have many values
- **required** - if attribute can be removed
- **type** - type of attribute values.

Default implementation provides two attribute types:

- **text** - java.lang.String object
- **binary** - byte[] object

## 27.1. Sample operations

```
Identity user = session.getPersistenceManager().
createIdentity("sampleUser");

// Check that binary attribute 'picture' is mapped

AttributeDescription attributeDescription =
session.getAttributesManager().
getAttributeDescription(user, "picture");
assertNotNull(attributeDescription);
assertEquals("binary", attributeDescription.getType());


// Generate random binary data for binary attribute

Random random = new Random();
byte[] picture = new byte[5120];
random.nextBytes(picture);
```

```
// User attributes
Attribute[] userInfo = new Attribute[]
{
new SimpleAttribute(P3PConstants.INFO_USER_NAME_GIVEN,
new String[]{"John"}),
new SimpleAttribute(P3PConstants.INFO_USER_NAME_FAMILY,
new String[]{"Doe"}),
new SimpleAttribute("picture", new byte[][]{picture})
};

session.getAttributesManager().
addAttributes(user, userInfo);

....

AttributesManager attrMgr = session.getAttributesManager();

attrMgr.addAttribute(anneUser,
P3PConstants.INFO_USER_NAME_GIVEN, "Anne");
attrMgr.addAttribute(anneUser,
P3PConstants.INFO_USER_NAME_FAMILY, "Smith");
attrMgr.addAttribute(anneUser,
P3PConstants.INFO_USER_JOB_TITLE, "Senior Software Developer");
attrMgr.addAttribute(anneUser,
P3PConstants.INFO_USER_BUSINESS_INFO_ONLINE_EMAIL, "anne.smith@acme.com");
attrMgr.addAttribute(anneUser,
P3PConstants.INFO_USER_BUSINESS_INFO_TELECOM_MOBILE_NUMBER, "777 777 777 7
 77");
```

# SPI

On the SPI level IdentityObject can be associated with several IdentityObjectAttribute objects. IdentityObjectAttribute is described by IdentityObjectAttributeMetaData object that contains its properties such as:

- **name** - attribute name
- **readonly** - if attribute values can be changed
- **multivalued** - if attribute can have many values
- **required** - if attribute can be removed
- **type** - type of attribute values.

IdentityObjectAttribute types supported by default implementations are the same as in the API level:

- **text** - java.lang.String object
- **binary** - byte[] object

All operations related to IdentityObjectAttribute are exposed by the AttributeStore interface

# Part VIII. Credentials

# API

API contains Credential and CredentialType interfaces. CredentialType defines type of credential object. Default implementation supports two types:

- **PASSWORD** - text password represented by java.lang.String object
- **BINARY** - binary credential represented by byte[]. For example some kind of certificate.

Two basic implementations are provided:

- **org.picketlink.idm.impl.api.BinaryCredential** - Credential with BINARY CredentialType
- **org.picketlink.idm.impl.api.PasswordCredential** - Credential with PASSWORD CredentialType

Because credentials values are stored as hash or in other encoded form both SPI and API only enables to update and validate credential value and not to read it from persistence store. API enables to only protect Identity objects with credentials. All related management operations are exposed in AttributesManager interface.

```
User anotherOne = session.getPersistenceManager().createUser("blah1");

session.getAttributesManager().updatePassword(anotherOne, "Password2000");
assertTrue(session.getAttributesManager().validatePassword(anotherOne, "Password2000"));

Credential password = new PasswordCredential("SuperPassword2345");
session.getAttributesManager().updateCredential(anotherOne, password);
assertTrue(session.getAttributesManager().validateCredentials(anotherOne, new
 Credential[]{password}));

// binary credential
byte[] cert = new byte[512000];
random.nextBytes(cert);
Credential binaryCredential = new BinaryCredential(cert);
session.getAttributesManager().updateCredential(anotherOne, binaryCredential);
assertTrue(session.getAttributesManager().validateCredentials(anotherOne, new
 Credential[]{binaryCredential}));
```

# SPI

SPI contains IdentityObjectCredential and IdentityObjectCredentialType interfaces that correspons to Credential and CredentialType interfaces in the API. Implementation supports the same two (PASSWORD and BINARY) types. Management operations are exposed in IdentityStore interface. Each IdentityObjectType can be configured to support different IdentityObjectCredentialType. This information is exposed by FeaturesMetaData interface.

IdentityObjectCredential interface exposes two methods to retrieve credential value:

- **getValue** - returns either String for text based credentials or byte[] for binary. True credential value may be needed by different IdentityStore implementations for validation. For example with LDAP authentication for IdentityObject entry will be performed
- **getEncodedValue** - Enables to provide IdentityStore with custom credential encoding method. IdentityStore is not obligated to use encoded value. This should return either String for text based credentials or byte[] for binary. May return null if credential implementation doesn't provide encoding mechanism. IdentityStore

# Part IX. Deployment

# IDM Usage Scenario

There are two ways to use the jboss idm:

- Use it as the embedded way
- Deploy it into the container (JBoss AS5), and then all other projects can use it by getting IdentitySessionFactory from JNDI.

For the 1st case, users need to use the API to start the IdentitySessionFactory, and then use it. The code is as following, which you can find on the example module.

```
IdentitySessionFactory identitySessionFactory = new IdentityConfigurationImpl().
   configure(new File("src/test/resources/example-db-config.xml")).buildIdentitySessionFactory();
IdentitySession identitySession =
 identitySessionFactory.createIdentitySession("realm::JBossIdentityExample_SampleRealm");
```

And then use the IdentitySession to do the operations etc. So it is very easy to use. We will look closer at the second scenario in the next chapter

# JBoss AS 5 Deployment

Now, lets look at the second case, by deploying the idm into the JBoss AS 5. By doing this different services can share the identity component, instead of having its own seperate identity component.

The jobs that need to be done for the deployment in the container is quite simple:

- Populate the idm schema if neccessary.
- Start the IdentitySessionFactory, and then register it into the JNDI.

Before we look at it further, lets see the configuration files that jboss idm needed typically. (Say using db back-end, hibernate impl combination)

- jboss idm configuration file. say jboss.idm.cfg.xml
- datasource file, say idm-ds.xml
- hibernate cfg file, say jboss.idm.hibernate.cfg.xml.

With regard to the detail of jboss idm configuration file, you can refer to the configuration documentation.

So, if we want to deploy the idm into container with a specified JNDI name, we need to have a deployment file to define the JNDI and other neccessary properties.

For the integration with JBoss AS5, the AS5 has a great deployment feature, we've built our own deployer to extend it, so that the AS can listen on the -jboss-idm.xml suffix file to start the IdentitySessionFactory.

Basically, we had two deployer, one is: IDMConfigParsingDeployer class, which is taking responsible for parsing files that ends with the -jboss-idm.xml suffix into Java object. The other is: IDMDeployer class, this one is to do the real job, which means it might populate the schema, initial dataset into target db, and then start the IdentitySessionFactory, register it into the JNDI with the specified name at last.

We will see a very typical deployment file looks like. (default-jboss-idm.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-idm-deployer xmlns="urn:jboss:identity:idm:deployer:v1_0_alpha"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="urn:jboss:identity:idm:deployer:v1_0_alpha identity-deployer.xsd">
   <JNDIName>java:/IdentitySessionFactory</JNDIName>
   <idmConfigFile>jboss.idm.cfg.xml</idmConfigFile>
```

```
    <hibernateDeployer>
     <hibernateConfiguration>jboss.idm.hibernate.cfg.xml</hibernateConfiguration>
<hibernateSessionFactoryJNDIName>java:/IDMHibernateSessionFactory</
hibernateSessionFactoryJNDIName>
    </hibernateDeployer>
    <initializers>
        <datasource>java:/jbossidmDS</datasource>
        <sqlInitializer>
            <sqlFile>idm-sql/jboss.idm.@database@.create.sql</sqlFile>
            <exitSQL>select * from jbid_io</exitSQL>
        </sqlInitializer>
    </initializers>
</jboss-idm-deployer>
```

- The deployment file must be named -jboss-idm.xml as suffix, otherwise, it won't be recoginzed in the JBoss AS5 container.
- The "JNDIName" and "idmConfigFile" attributes are required. The JNDIName is the name for keeping the started IdentitySessionFactory.
- The hibernateDeployer is optional, the reason that why we had the hibernateDeployer is that we can reuse the hibernateSessionFactory in the jboss idm configuration file.
- The Initializer is optional, it is responsible for populating the db schema and initialized dataset if any.

detailed information about the deployment file is specified in the identity-deployer.xsd file.

Once you've deployed the idm into JBoss AS5, by using the distribution. It will copy the idm-deployer into the JBoss AS5/server/$config/deployers folder, and the idm folder into the JBoss AS5/server/$config/deploy folder, which contains the default configuration files, like the jboss.idm.cfg.xml, idm-ds.xml etc.