PicketLink

PicketLink Reference Documentation

PicketLink

PicketLink PicketLink Reference Documentation PicketLink

Author

PicketLink is an umbrella project for security and identity management for Java Applications. PicketLink is an important project under the security offerings from JBoss.

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the plus sign that connects each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to a virtual terminal.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click

[1] https://fedorahosted.org/liberation-fonts/

> **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.
>
> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                 struct kvm_assigned_pci_dev *assigned_dev)
{
         int r = 0;
         struct kvm_assigned_dev_kernel *match;
```

```
        mutex_lock(&kvm->lock);

        match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
                                      assigned_dev->assigned_dev_id);
        if (!match) {
                printk(KERN_INFO "%s: device hasn't been assigned before, "
                  "so cannot be deassigned\n", __func__);
                r = -EINVAL;
                goto out;
        }

        kvm_deassign_device(kvm, match);

        kvm_free_assigned_device(kvm, match);

out:
        mutex_unlock(&kvm->lock);
        return r;
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

If you experience difficulty with a procedure described in this documentation, visit the PicketLink Issue Tracker at `https://issues.jboss.org/browse/PLINK`.

PicketLink's user forums should be your first port of call for any user-related issues with PicketLink. Be it a question, wanting to report a bug, looking for advice, or simply sharing some cool stories you may have. You can visit the user forums at `https://community.jboss.org/en/picketlink`.

Overview

1.1. The Top 8 Java Application Security Problems Solved by PicketLink

Are you developing a multiuser Java EE application? Do you find yourself asking:

1. **`What's the best way to add security to the application?`**

   PicketLink provides an easy way to enable security to an application. With a minimal configuration you are able to authenticate(*Section 2.1, "Overview"*), authorize(*Section 11.1, "Overview"*) and perform identity management(*Section 3.1, "Introduction"*) operations using a database or a LDAP identity store.

2. **`How do I authenticate and authorize users?`**

   If you're already faimilar with JBoss Seam 2, you'll find PicketLink very familiar. PicketLink provides a **`Identity`** bean(*Section 2.2, "Authentication API - The Identity Bean"*) to represent your users with useful methods for authentication, logout and authorization. PicketLink also provides a bunch of authorization annotations(*Section 11.1, "Overview"*) that you can use to easily protect your beans and their methods.

3. **`How do I add Identity and Access Management(IAM) to my application ?`**

   Identity Management is one of the core features provided by PicketLink. With a minimal setup you are able to manage users, roles, groups and relationships between them. It supports identity partitioning, useful for multi-tenancy architectures. You can even store your identity data using different repositories such as databases or LDAP servers.

4. **`How can I create a secure multi-tenancy architecture for my SaaS (Software as a Service) application ?`**

   PicketLink supports identity partitioning. This means that you can logically separate your identity data into partitions. You can use a single repository or even use a different repository for each partition. For instance, using different database servers for each partition.

5. **`How can I enable Single Sign-On for my applications?`**

   PicketLink provides an easy way to configure Single Sign-On based on the SAML specification. Enable your application as an Identity Provider or Service Provider requires only a few configuration.

6. **`How do I add authentication and authorization to my REST layer and API ?`**

   PicketLink provides several features to secure REST-based applications. Regarding authentication, it supports both stateful and stateless authentication models, depending on your requirements. When using a steteless model user is re-authenticated on every single request, ideal when you have a huge load of requests. It also provides a simple API to manage JSON Web Token(JWT) and Javascript Object Signing and Encryption(JOSE), which you can use to create your own tokens or consume tokens following the format defined by these specifications. Authorization can also be done by using the different annotations provided by PicketLink. You can even define your own authorization annotations.

7. **`How do I protect my application HTTP resources and pages ?`**

PicketLink provides an easy, simple and powerful API for Http Security fully integrated with the Java Servlet API. It allows you to configure your aplication resources or paths to enable authentication and authorization. With a minimal setup you can enforce security policies such as authentication and authorization based on roles, groups, realms or even use EL expressions.

8. **`How can my application authenticate users using their Facebook, Twitter, or Google accounts ?`**

   PicketLink provides support for Social Authentication using the most common providers such as Facebook, Twitter and Google. Please, take a look at the quickstarts(*Section 1.3.1, "QuickStarts"*) for some examples.

If these are your questions, then PicketLink is your answer. If not, you're welcome to contribute with ideas or code to improve PicketLink offerings.

Report a bug[1]

1.2. What is PicketLink?

PicketLink is an Application Security Framework for Java EE applications. It provides features for authenticating users, authorizing access to the business methods of your application, managing your application's users, groups, roles and permissions, plus much more. The following diagram presents a high level overview of the PicketLink modules and the main features provided by those modules:

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+31227-700763+%5BLatest%5D&comment=Title%3A+The+Top+8+Java+Application+Security+Problems+Solved+by+PicketLink%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=31227-700763+21+Aug+2014+01%3A22+en-US+%5BLatest%5D

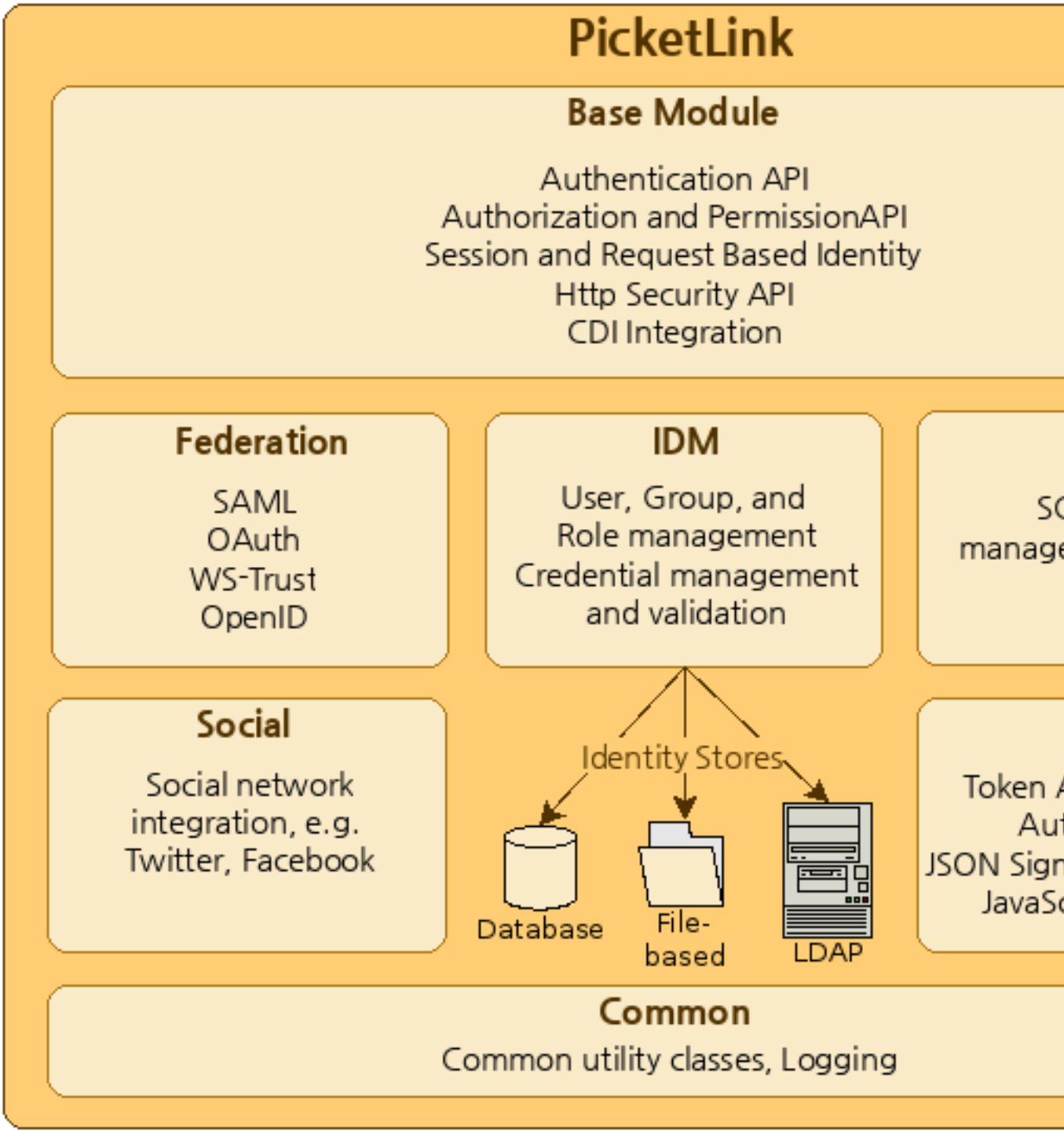


Figure 1.1. PicketLink Overview

Report a bug[2]

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29653-640996+%5BLatest%5D&comment=Title%3A+What+is+PicketLink%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29653-640996+15+May+2014+10%3A07+en-US+%5BLatest%5D

1.3. Where do I get started?

Depending on exactly which PicketLink features you'd like to use, getting started can be as simple as adding the PicketLink jar libraries to your project (see *Section 1.6, "Maven Dependencies"* below for more info) and writing a few lines of code.

To get started using *PicketLink Identity Management* to manage the users and other identity objects in your application, you can head straight to *Section 3.2, "Getting Started - The 5 Minute Guide"*.

If you don't wish to use PicketLink IDM but would like to use PicketLink for authentication in your Java EE application but *control the authentication process yourself* then head to *Section 2.3.1, "A Basic Authenticator"* for simplistic example which you may adapt for your own application.

If you want to take a look on how to enable authorization to your application then head to *Section 11.1, "Overview"*.

If you want to take a look on how to enable authentication and authorization to your application HTTP resources and pages then head to TODO.

If you wish to use *SAML SSO* then you can head to *Chapter 14, Federation*.

For new users, we recommend the following steps for a better and speed learning of PicketLink:

- **`Download JBoss EAP or WildFly`**

  PicketLink can be used on both servers. Use the *Section 1.7, "PicketLink Installer"* to configure them with the latest version of the PicketLink modules and libraries.

- **`Get the Quickstarts Up and Running`**

  This is a very good way to learn from examples. Use your favorite IDE to import the *Section 1.3.1, "QuickStarts"* and look at their sources. All quickstarts can be deployed on both JBoss EAP and WildFly.

- **`Spend Some Time Reading the Available Guides`**

  Each one cover an important PicketLink aspect. They explain some core concepts and make easier to understand the quickstarts. All guides are available at PicketLink Oficial Site at *www.picketlink.org/*[3].

- **`Read the Reference Documentation`**

  You're already doing this, right ?

- **`PicketLink JavaDoc Reference`**

  For more detailed information on JavaDoc for API of PicketLink. See *http://docs.jboss.org/picketlink/2/latest/api/*

- **`Use the User Forum`**

  And post your questions and feedbacks. See *https://community.jboss.org/en/picketlink/content?filterID=contentstatus[published]~objecttype~objecttype[thread]*

Here's some additional resources also to help you get started:

[3] http://www.picketlink.org/gettingstarted/

Report a bug[4]

1.3.1. QuickStarts

Please head to *Chapter 15, PicketLink Quickstarts* for more information about our quickstarts, covering some useful usecases and most of PicketLink features.

Report a bug[5]

1.3.2. API Documentation

The latest version of the PicketLink API documentation can be found at *http://docs.jboss.org/picketlink/2/latest/api/*. This handy reference describes the user-facing classes and methods provided by the PicketLink libraries.

Report a bug[6]

1.4. Modules

1.4.1. Base module

The base module provides the integration framework required to use PicketLink within a Java EE application. It defines a flexible authentication API that allows pluggable authentication mechanisms to be easy configured, with a sensible default authentication policy that delegates to the identity management subsystem. It provides session-scoped authentication tracking for web applications and other session-capable clients, plus a customisable permissions SPI that supports a flexible range of authorization mechanisms for object-level security. It is the "glue" that integrates all of the PicketLink modules together to provide a cohesive API, and also provides CDI producers to allow you to inject the PicketLink API objects directly into your application's beans.

The base module libraries are as follows:

- **`picketlink-api`** - API for PicketLink's base module.
- **`picketlink-impl`** - Internal implementation classes for the base API.

Report a bug[7]

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29654-700764+%5BLatest%5D&comment=Title%3A+Where+do+I+get+started%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29655-620652+%5BLatest%5D&comment=Title%3A+QuickStarts%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29655-620652+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29656-621122+%5BLatest%5D&comment=Title%3A+API+Documentation%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29656-621122+13+Mar+2014+11%3A08+en-US+%5BLatest%5D

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29657-620709+%5BLatest%5D&comment=Title%3A+Base+module%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29657-620709+12+Mar+2014+12%3A44+en-US+%5BLatest%5D

1.4.2. Identity Management

The Identity Management module defines the base identity model; a collection of interfaces and classes that represent the identity constructs (such as users, groups and roles) used throughout PicketLink (see the Identity Management chapter for more details). As such, it is a required module and must always be included in any application deployments that use PicketLink for security. It also provides a uniform API for managing the identity objects within your application. The Identity Management module has been designed with minimal dependencies and may be used in a Java SE environment, however the recommended environment is Java EE in conjunction with the base module.

Libraries are as follows:

- **`picketlink-idm-api`** - PicketLink's Identity Management (IDM) API. This library defines the Identity Model central to all of PicketLink, and all of the identity management-related interfaces.
- **`picketlink-idm-impl`** - Internal implementation classes for the IDM API.

Report a bug[8]

1.4.3. Authorization Module

The Authorization Module provides a flexible authorization API that allows you to easily protected your beans and their methods by using a set of security annotations. Those annotations provide some built-in authorization checks based on:

- Roles
- Groups
- Partitions
- Authenticated User
- EL-based expressions
- Permissions

All authorization is performed based on the information stored by PicketLink IDM.

This module is based on Apache DeltaSpike Security module, which means that DeltaSpike's dependencies are transitive and must be available in your project's classpath. Usually, these dependencies would be automatically added to your project.

The authorization module libraries is:

- **`picketlink-deltaspike`**

Report a bug[9]

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29658-620264+%5BLatest%5D&comment=Title%3A+Identity+Management%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29658-620264+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35785-665366+%5BLatest%5D&comment=Title%3A+Authorization+Module%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35785-665366+07+Jun+2014+08%3A54+en-US+%5BLatest%5D

1.4.4. JSON

Error

Topic 35786 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 35786* for more detailed information.

Report a bug[10]

1.4.5. Federation

The Federation module is an optional module that implements a number of Federated Identity standards, such as SAML (both version 1.1 and 2.0), WS-Trust and OpenID.

Report a bug[11]

1.5. License

PicketLink is licensed under the Apache License Version 2, the terms and conditions of which can be found at *apache.org*[12].

Report a bug[13]

1.6. Maven Dependencies

The PicketLink libraries are available from Maven Central Repository. The dependencies can be easily configured in your Maven-based project by using the PicketLink Bill of Materials(BOM). A BOM is a very handy tool to properly manage your dependencies and their versions keeping your project in sync with the libraries supported and distributed by a specific PicketLink version.

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35786-701173+%5BLatest%5D&comment=Title%3A+JSON%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35786-701173+22+Aug+2014+02%3A11+en-US+%5BLatest%5D

[11] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29659-620265+%5BLatest%5D&comment=Title%3A+Federation%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29659-620265+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

[12] http://www.apache.org/licenses/LICENSE-2.0.html

[13] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29660-626685+%5BLatest%5D&comment=Title%3A+License%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29660-626685+01+Apr+2014+16%3A02+en-US+%5BLatest%5D

Note

PicketLink libraries are also available on JBoss Release Repository at *https://repository.jboss.org/nexus/index.html#nexus-search;gav~org.picketlink~~2.7.0.CR1~~* and Maven Central at *http://search.maven.org/#search%7Cga%7C1%7Cg%3A%22org.picketlink%22%20AND%20v%3A%222.7.0.CR1%22*.

For most applications using Java EE 6.0 and beyond, the *picketlink-javaee-6.0* BOM can be used to define the PicketLink and Java EE 6.0 dependencies to your project.

```
<properties>
  <!-- PicketLink dependency versions -->
  <version.picketlink.javaee.bom>
2.7.0.CR1</version.picketlink.javaee.bom>
</properties>

<dependencyManagement>
  <dependencies>
    <!-- Dependency Management for PicketLink and Java EE 6.0. -->
    <dependency>
      <groupId>org.picketlink</groupId>
      <artifactId>picketlink-javaee-6.0</artifactId>
      <version>${version.picketlink.javaee.bom}</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Once the BOM is properly configured, you need to configure the PicketLink dependencies. When using a BOM you don't need to specify their versions because this is managed automatically, the version in use depends on the BOM's version. For example, the configuration above is defining a version 2.7.0.CR1 of the *picketlink-javaee-6.0* BOM, which means you'll be using version 2.7.0.CR1 of PicketLink libraries.

For a typical application, it is suggested that you include the following PicketLink dependency:

```
<dependencies>
    <!--
        PicketLink Uber Dependency. It provides all PicketLink dependencies from a single
 JAR. You still can define each module separately, if you want to.
    -->
  <dependency>
    <groupId>org.picketlink</groupId>
    <artifactId>picketlink</artifactId>
  </dependency>
</dependencies>
```

The *org.picketlink:picketlink* dependency above is a *uber jar* containing the most common PicketLink modules and their dependencies.

- **`picketlink-api`** - API for PicketLink's Base module.
- **`picketlink-impl`** - Internal implementation classes for the Base API.
- **`picketlink-idm-api`** - API for PicketLink's IDM module.

- **`picketlink-idm-impl`** - Internal implementation classes for the IDM API.
- **`picketlink-deltaspike`** - API for authorization, providing some built-in annotations. This module is based on Apache DeltaSpike Security.

This is by far the most easy way to configure PicketLink in your project. The *org.picketlink:picketlink* dependency avoids you to specify each module separately.It provides to your project all *Authentication*, *Authorization* and *Identity Management* features provided by PicketLink.

However, you may not use the *org.picketlink:picketlink* dependency at all and define each module in your project's pom.xml. The minimal dependencies required to enable PicketLink in your project are:

```
<dependencies>
<!-- Import the PicketLink API, we deploy this as library with the application,
        and can compile against it -->
  <dependency>
    <groupId>org.picketlink</groupId>
    <artifactId>picketlink-api</artifactId>
  </dependency>

<!-- Import the PicketLink Implementation, we deploy this as library with the application,
        but can't compile against it -->
  <dependency>
    <groupId>org.picketlink</groupId>
    <artifactId>picketlink-impl</artifactId>
    <scope>runtime</scope>
  </dependency>

</dependencies>
```

The dependencies above provide to your project all *Authentication* and *Identity Management* features provided by PicketLink.

If you also want to enable *Authorization* to your project, you can define the *org.picketlink:picketlink-deltaspike* dependency as follows:

```
<dependencies>
  <dependency>
    <groupId>org.picketlink</groupId>
    <artifactId>picketlink-deltaspike</artifactId>
  </dependency>
</dependencies>
```

Note

The authorization features provided by PicketLink are based on Apache DeltaSpike. When you configure the *org.picketlink:picketlink-deltaspike* dependency to your project, Deltaspike's dependencies are added to your project's classpath and shipped within your application.

If you want to configure the PicketLink Identity Management dependencies, then add the following to your project's pom.xml:

```
<dependencies>
  <dependency>
      <groupId>org.picketlink</groupId>
```

```
    <artifactId>picketlink-idm-api</artifactId>
    <scope>compile</scope>
</dependency>

<dependency>
    <groupId>org.picketlink</groupId>
    <artifactId>picketlink-idm-impl</artifactId>
    <scope>runtime</scope>
</dependency>
```

Note

We strongly recommend using a BOM to configure your project with the PicketLink dependencies. This can avoid some very common and tricky issues like keep the dependencies in sync with the versions distributed in a release.

Report a bug[14]

1.7. PicketLink Installer

The PicketLink Installer is a simple Apache Ant script that applies all the necessary changes to your JBoss Enterprise Application Platform or WildFly installation, including:

- Updates the PicketLink module with the latest libraries.

Important

The installer is not a required step in order to get you started with PicketLink. But if you want the PicketLink module updated with a specific version(in order to avoid ship the libraries inside your deployment) in your server installation, it can be very useful.

The installer can be obtained from *http://downloads.jboss.org/picketlink/2/2.7.0.CR1/picketlink-installer-2.7.0.CR1.zip*. Once you've downloaded, extract the ZIP file, enter the directory that was created and execute the following command:

```
ant
```

Now you should be prompted for the full path of your JBoss Application Server installation.

```
prepare:
     [echo]
     [echo]
 ###############################################################################
```

[14] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29661-700765+%5BLatest%5D&comment=Title%3A+Maven+Dependencies%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29661-700765+21+Aug+2014+01%3A42+en-US+%5BLatest%5D

```
     [echo]       Welcome to the PicketLink Installer
     [echo]
     [echo]       This installer will update your JBoss Enterprise Application Platform
 installation with the
     [echo]       following libraries and their dependencies:
     [echo]
     [echo]       - PicketLink Core
 2.7.0.CR1
     [echo]       - PicketLink Identity Management
 2.7.0.CR1
     [echo]       - PicketLink Federation
 2.7.0.CR1
     [echo]
     [echo]       New modules will be added to your installation.
     [echo]
 ###############################################################################
     [echo]
    [input] Which JBoss Application Server are you using ? ([eap], wildfly)
    eap
    [input] Please enter the path to your JBoss Application Server installation:
```

And it is done !

Note

If you're using JBoss EAP, make sure you're using version 6.2+.

Report a bug[15]

1.8. Referencing PicketLink from JBoss Modules

All the necessary PicketLink libraries are available in your JBoss Application Server installation from JBoss Modules. To configure them in your deployment, just add a **`META-INF/jboss-deployment-structure.xml`** file inside the root directory of your deployment to configure the dependencies as follows:

```
<jboss-deployment-structure>
 <deployment>
  <dependencies>
   <!-- This will enable PicketLink Federation to your deployment. -->
   <module name="org.picketlink" />
  </dependencies>
 </deployment>
</jboss-deployment-structure>
```

```
<jboss-deployment-structure>
 <deployment>
  <dependencies>
   <!-- This will enable PicketLink Authentication/Authorization and IDM dependencies to your
 deployment. -->
```

[15] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29662-700767+%5BLatest%5D&comment=Title%3A+PicketLink+Installer%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29662-700767+21+Aug+2014+01%3A47+en-US+%5BLatest%5D

```
    <module name="org.picketlink.core" meta-inf="import"/>
    <module name="org.picketlink.core.api" meta-inf="import"/>
    <module name="org.picketlink.idm.api" meta-inf="import"/>
   </dependencies>
  </deployment>
</jboss-deployment-structure>
```

```
<jboss-deployment-structure>
 <deployment>
  <dependencies>
   <!-- This will enable only the IDM dependencies to your deployment. -->
   <module name="org.picketlink.idm" />
  </dependencies>
 </deployment>
</jboss-deployment-structure>
```

It is strongly recommended that you use the PicketLink libraries from your JBoss Application Server modules. You don't need to add any additional library to your deployments and you can easily manage the PicketLink libraries without requiring changes to your deployments.

Considering that you no longer need the PicketLink libraries inside your deployment, you must change your Maven dependencies to use the PicketLink dependencies with scope **`provided`**:

```
<dependency>
 <groupId>org.picketlink</groupId>
 <artifactId>picketlink-api</artifactId>
 <scope>provided</scope>
</dependency>

<dependency>
 <groupId>org.picketlink</groupId>
 <artifactId>picketlink-idm-api</artifactId>
 <scope>provided</scope>
</dependency>
```

Report a bug[16]

1.9. Help us improve the docs!

We're always looking for ways to improve this documentation. If you think that we can enhance the way that any of PicketLink's features or concepts are explained, or even if you just spot a typo or grammatical error then please let us know on the PicketLink forums (you can find a link to the forums at *www.picketlink.org*[17]). We appreciate any and all feedback that is provided.

Report a bug[18]

[16] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+39276-676191+%5BLatest%5D&comment=Title%3A+Referencing+PicketLink+from+JBoss+Modules%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=39276-676191+24+Jun+2014+22%3A41+en-US+%5BLatest%5D

[17] http://www.picketlink.org

[18] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29663-626687+%5BLatest%5D&comment=Title%3A+Help+us+improve+the+docs%21%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29663-626687+01+Apr+2014+16%3A02+en-US+%5BLatest%5D

Authentication

2.1. Overview

Authentication is the act of verifying the identity of a user. PicketLink offers an extensible authentication API that allows for significant customization of the authentication process, while also providing sensible defaults for developers that wish to get up and running quickly. It also supports both synchronous and asynchronous user authentication, allowing for both a traditional style of authentication (such as logging in with a username and password), or alternatively allowing authentication via a federated identity service, such as OpenID, SAML or OAuth. This chapter will endeavour to describe the authentication API and the authentication process in some detail, and is a good place to gain a general overall understanding of authentication in PicketLink. However, please note that since authentication is a cross-cutting concern, various aspects (for example Identity Management-based authentication and Federated authentication) are documented in other chapters of this book.

Report a bug[1]

2.2. Authentication API - The Identity Bean

The **`Identity`** bean (which can be found in the **`org.picketlink`** package) is central to PicketLink's security API. This bean represents the authenticated user for the current session, and provides many useful methods for controlling the authentication process and querying the user's assigned privileges. In terms of authentication, the **`Identity`** bean provides the following methods:

```
AuthenticationResult login();

void logout();

boolean isLoggedIn();

Account getAccount();
```

The **`login()`** method is the *primary* point of entry for the authentication process. Invoking this method will cause PicketLink to attempt to authenticate the user based on the credentials that they have provided. The **`AuthenticationResult`** type returned by the **`login()`** method is a simple enum that defines the following two values:

```
public enum AuthenticationResult {
    SUCCESS, FAILED
}
```

If the authentication process is successful, the **`login()`** method returns a result of **`SUCCESS`**, otherwise it returns a result of **`FAILED`**. The default implementation of the **`Identity`** bean is a **`@SessionScoped`** CDI bean, which means that once a user is authenticated they will stay authenticated for the duration of the session.

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29664-620270+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29664-620270+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

Note

One significant point to note is the presence of the **@Named** annotation on the **Identity** bean, which means that its methods may be invoked directly from the view layer (if the view layer, such as JSF, supports it) via an EL expression.

One possible way to control the authentication process is by using an action bean, for example the following code might be used in a JSF application:

```
public @RequestScoped @Named class LoginAction {

    @Inject Identity identity;

    public void login() {
        AuthenticationResult result = identity.login();
        if (AuthenticationResult.FAILED.equals(result)) {
            FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage(
                        "Authentication was unsuccessful.  Please check your username and
 password " +
                        "before trying again."));
        }
    }
}
```

In the above code, the **Identity** bean is injected into the action bean via the CDI **@Inject** annotation. The **login()** method is essentially a wrapper method that delegates to **Identity.login()** and stores the authentication result in a variable. If authentication was unsuccessful, a **FacesMessage** is created to let the user know that their login failed. Also, since the bean is **@Named** it can be invoked directly from a JSF control like so:

```
<h:commandButton value="LOGIN" action="#{loginAction.login}"/>
```

The **Identity.isLoggedIn()** method may be used to determine whether there is a user logged in for the current session. It is typically used as an authorization check to control either an aspect of the user interface (for example, not displaying a menu item if the user isn't logged in), or to restrict certain business logic. While logged in, the **getAccount()** method can be used to retrieve the currently authenticated account (i.e. the user). If the current session is not authenticated, then **getAccount()** will return **null**. The following example shows both the **isLoggedIn()** and **getAgent()** methods being used inside a JSF page:

```
<ui:fragment rendered="#{identity.loggedIn}">Welcome, #{identity.account.loginName}
```

Note

If you're wondering what an **Account** is, it is simply a representation of the external entity that is interacting and authenticating with your application. The **Account** interface is actually the superclass of the **User** and **Agent** - see the Identity Management chapter for more details.

The **`logout()`** method allows the user to log out, thereby clearing the authentication state for their session. Also, if the user's session expires (for example due to inactivity) their authentication state will also be lost requiring the user to authenticate again.

The following JSF code example demonstrates how to render a log out button when the current user is logged in:

```
<ui:fragment rendered="#{identity.loggedIn}">
    <h:form>
        <h:commandButton value="Log out" action="#{identity.logout}"/>
    </h:form>
</ui:fragment>
```

While it is the **`Identity`** bean that controls the overall authentication process, the actual authentication "business logic" is defined by the **`Authenticator`** interface:

```
public interface Authenticator {
    public enum AuthenticationStatus {
        SUCCESS,
        FAILURE,
        DEFERRED
    }

    void authenticate();

    void postAuthenticate();

    AuthenticationStatus getStatus();

    Account getAccount();
}
```

During the authentication process, the **`Identity`** bean will invoke the methods of the *active* **`Authenticator`** (more on this in a moment) to perform user authentication. The **`authenticate()`** method is the most important of these, as it defines the actual authentication logic. After **`authenticate()`** has been invoked by the **`Identity`** bean, the **`getStatus()`** method will reflect the authentication status (either **`SUCCESS`**, **`FAILURE`** or **`DEFERRED`**). If the authentication process was a success, the **`getAccount()`** method will return the authenticated **`Account`** object and the **`postAuthenticate()`** method will be invoked also. If the authentication was not a success, **`getAccount()`** will return **`null`**.

Note

By default, the **`Identity`** bean uses a **`Authenticator`** fully integrated with the Identity Management API. You don't need to provide your own authenticator to start using PicketLink! Check *Section 3.1, "Introduction"* for more details about how to manage users and credentials.

Report a bug[2]

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29665-700768+%5BLatest%5D&comment=Title%3A+Authentication+API+-+The+Identity+Bean%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

2.2.1. Stateful or Stateless Authentication

The **`Identity`** bean is stateful by default, using the session scope to share user's authentication state between different requests and interactions. Once the user is authenticated, the **`Identity`** bean will hold user's data until the session is invalidated.

When you inject the **`Identity`** bean as follows:

```
@Inject
private Identity identity;
```

You're using the stateful version of the **`Identity`** bean.

While this is the desired behavior for most applications, this may not be the case if you're exposing the **`Identity`** bean as a service, with a high load of authentication requests. A good example is a RESTful endpoint that issues a token to represent user's information after a successful authentication, using that token in subsequent invocations to other services to check if the user was previously authenticated, check the token validity or even provide some authorization. Pretty common scenario for mobile and HTML5 applications.

In those cases, you're not interested in wasting server resources to manage users sessions. You want a stateless behavior, where once the authentication finishes all data produced during the process is lost after the request processing is done.

To configure the **`Identity`** bean as stateless you need to provide the following configuration:

```
public class SecurityConfiguration {

    public void init(@Observes SecurityConfigurationEvent event) {
        SecurityConfigurationBuilder securityConfigurationBuilder = event.getBuilder();

        securityConfigurationBuilder
            .identity()
                .stateless(); // enable stateless authentication
    }
}
```

The **`org.picketlink.event.SecurityConfigurationEvent`** is an event that lets you configure PicketLink. It provides access to a **`org.picketlink.config.SecurityConfigurationBuilder`** instance from where you can provide all configuration for PicketLink.

Report a bug[3]

2.2.2. Defining a Custom Scope

The **`Identity`** bean can be also configured to use a specific scope, depending on your requirements. For example, Apache DeltaSpike features a **`WindowScoped`**, a normal passivating scope. It makes sense to want to have a window scoped identity to enable a user-per-window application (like Google does).

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30301-700770+%5BLatest%5D&comment=Title%3A+Stateful+or+Stateless+Authentication%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30301-700770+21+Aug+2014+02%3A53+en-US+%5BLatest%5D

In order to specify the **`Identity`** bean scope you just need a configuration as follows:

```
public class SecurityConfiguration {

    public void init(@Observes SecurityConfigurationEvent event) {
        SecurityConfigurationBuilder securityConfigurationBuilder = event.getBuilder();

        securityConfigurationBuilder
            .identity()
                .scope(WindowScoped.class);
    }
}
```

Note

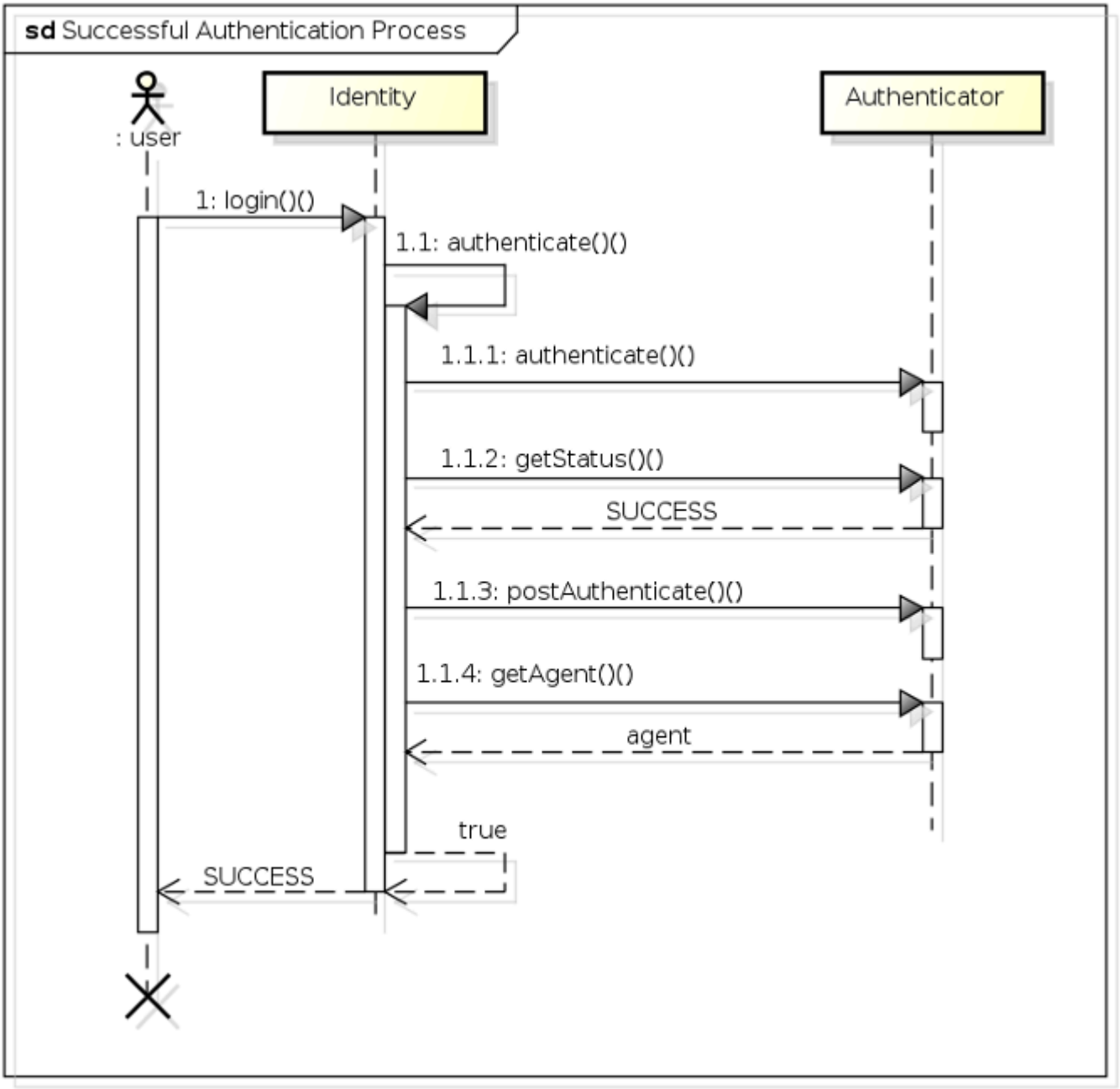
When specifying a scope make sure it is a **`NormalScope`** and not a pseudo-scope.

Report a bug[4]

2.3. The Authentication Process

Now that we've looked at all the individual pieces, let's take a look at how they all work together to process an authentication request. For starters, the following sequence diagram shows the class interaction that occurs during a successful authentication:

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41720-708286+%5BLatest%5D&comment=Title%3A+Defining+a+Custom+Scope%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41720-708286+11+Sep+2014+05%3A45+en-US+%5BLatest%5D



- *1* - The user invokes the **`login()`** method of the **`Identity`** bean.
- *1.1* - The **`Identity`** bean (after performing a couple of validations) invokes its own **`authenticate()`** method.
- *1.1.1* - Next the **`Identity`** bean invokes the **`Authenticator`** bean's **`authenticate()`** method (which has a return value of **`void`**).
- *1.1.2* - To determine whether authentication was successful, the **`Identity`** bean invokes the **`Authenticator`**'s **`getStatus()`** method, which returns a **`SUCCESS`**.
- *1.1.3* - Upon a successful authentication, the **`Identity`** bean then invokes the **`Authenticator`**'s **`postAuthenticate()`** method to perform any post-authentication logic.

- *1.1.4* - The **`Identity`** bean then invokes the **`Authenticator`**'s **`getAccount()`** method, which returns an **`Account`** object representing the authenticated agent, which is then stored as a private field in the **`Identity`** bean.

The authentication process ends when the **`Identity.authenticate()`** method returns a value of **`true`** to the **`login()`** method, which in turn returns an authentication result of **`SUCCESS`** to the invoking user.

Report a bug[5]

2.3.1. A Basic Authenticator

Let's take a closer look at an extremely simple example of an **`Authenticator`**. The following code demonstrates an **`Authenticator`** implementation that simply tests the username and password credentials that the user has provided against hard coded values of **`jsmith`** for the username, and **`abc123`** for the password, and if they match then authentication is deemed to be a success:

```
@PicketLink
public class SimpleAuthenticator extends BaseAuthenticator {

    @Inject DefaultLoginCredentials credentials;

    @Override
    public void authenticate() {
        if ("jsmith".equals(credentials.getUserId()) &&
                "abc123".equals(credentials.getPassword())) {
            setStatus(AuthenticationStatus.SUCCESS);
            setAccount(new User("jsmith"));
        } else {
            setStatus(AuthenticationStatus.FAILURE);
            FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(
                    "Authentication Failure - The username or password you provided were
 invalid."));
        }
    }
}
```

The first thing we can notice about the above code is that the class is annotated with the **`@PicketLink`** annotation. This annotation indicates that this bean should be used for the authentication process. The next thing is that the authenticator class extends something called **`BaseAuthenticator`**. This abstract base class provided by PicketLink implements the **`Authenticator`** interface and provides implementations of the **`getStatus()`** and **`getAccount()`** methods (while also providing matching **`setStatus()`** and **`setAccount()`** methods), and also provides an empty implementation of the **`postAuthenticate()`** method. By extending **`BaseAuthenticator`**, our **`Authenticator`** implementation simply needs to implement the **`authenticate()`** method itself.

We can see in the above code that in the case of a successful authentication, the **`setStatus()`** method is used to set the authentication status to **`SUCCESS`**, and the **`setAccount()`** method is used to set the user (in this case by creating a new instance of **`User`**). For an unsuccessful authentication, the **`setStatus()`** method is used to set the authentication status to **`FAILURE`**, and a new **`FacesMessage`** is created to indicate to the user that authentication has failed. While this code

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29666-675001+%5BLatest%5D&comment=Title%3A+The+Authentication+Process%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

is obviously meant for a JSF application, it's possible to execute whichever suitable business logic is required for the view layer technology being used.

One thing that hasn't been touched on yet is the following line of code:

```
@Inject DefaultLoginCredentials credentials;
```

This line of code injects the credentials that have been provided by the user using CDI's **`@Inject`** annotation, so that our **`Authenticator`** implementation can query the credential values to determine whether they're valid or not. We'll take a look at credentials in more detail in the next section.

Note

You may be wondering what happens if you don't provide an **`Authenticator`** bean in your application. If this is the case, PicketLink will automatically authenticate via the identity management API, using a sensible default configuration. See the Identity Management chapter for more information.

Report a bug[6]

2.3.2. Multiple Authenticator Support

If your application requires multiple authentication methods, you can use multiple **`Authenticator`** implementations and decide which one should be used depending on some decision logic. For that, use a producer method annotated with **`@PicketLink`** to produce the desired **`Authenticator`**:

```
@RequestScoped
@Named
public class AuthenticatorSelector {

    @Inject Instance<CustomAuthenticator> customAuthenticator;
    @Inject Instance<IdmAuthenticator> idmAuthenticator;

    private String authenticator;

    public String getAuthenticator() {
        return authenticator;
    }

    public void setAuthenticator(String authenticator) {
        this.authenticator = authenticator;
    }

    @Produces
    @PicketLink
    public Authenticator selectAuthenticator() {
        if ("custom".equals(authenticator)) {
            return customAuthenticator.get();
```

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29667-620273+%5BLatest%5D&comment=Title%3A+A+Basic+Authenticator%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29667-620273+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

```
        } else {
            return idmAuthenticator.get();
        }
    }
}
```

This **@Named** bean exposes an **authenticationSelector.authenticator** property that can be set by a user interface control in the view layer. If its value is set to "custom" then **CustomAuthenticator** will be used, otherwise **IdmAuthenticator** (the **Authenticator** used to authenticate using the Identity Management API) will be used instead. Another good example is when you need to choose the **Authenticator** based on a request header or parameter value.

Report a bug[7]

2.3.3. Credentials

Credentials are something that provides evidence of a user's identity; for example a username and password, an X509 certificate or some kind of biometric data such as a fingerprint. PicketLink has extensive support for a variety of credential types, and also makes it relatively simple to add custom support for credential types that PicketLink doesn't support out of the box itself.

In the previous section, we saw a code example in which a **DefaultLoginCredentials** (an implementation of the **Credentials** interface that supports a user ID and a credential value) was injected into the **SimpleAuthenticator** bean. The most important thing to know about the **Credentials** interface in relation to writing your own custom **Authenticator** implementation is that *you're not forced to use it*. *However*, while the **Credentials** interface is mainly designed for use with the Identity Management API (which is documented in a separate chapter) and its methods would rarely be used in a custom **Authenticator**, PicketLink provides some implementations which are suitably convenient to use as such, **DefaultLoginCredentials** being one of them.

So, in a custom **Authenticator** such as this:

```
public class SimpleAuthenticator extends BaseAuthenticator {

    @Inject DefaultLoginCredentials credentials;

    // code snipped
}
```

The credential injection is totally optional. As an alternative example, it is totally valid to create a request-scoped bean called **UsernamePassword** with simple getters and setters like so:

```
public @RequestScoped class UsernamePassword {
    private String username;
    private String password;

    public String getUsername() { return username; }
    public String getPassword() { return password; }

    public void setUsername(String username) { this.username = username; }
```

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29668-627120+%5BLatest%5D&comment=Title%3A+Multiple+Authenticator+Support%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29668-627120+03+Apr+2014+03%3A33+en-US+%5BLatest%5D

```
    public void setPassword(String password) { this.password = password; }
}
```

And then inject that into the **`Authenticator`** bean instead:

```
public class SimpleAuthenticator extends BaseAuthenticator {

    @Inject UsernamePassword usernamePassword;

    // code snipped
}
```

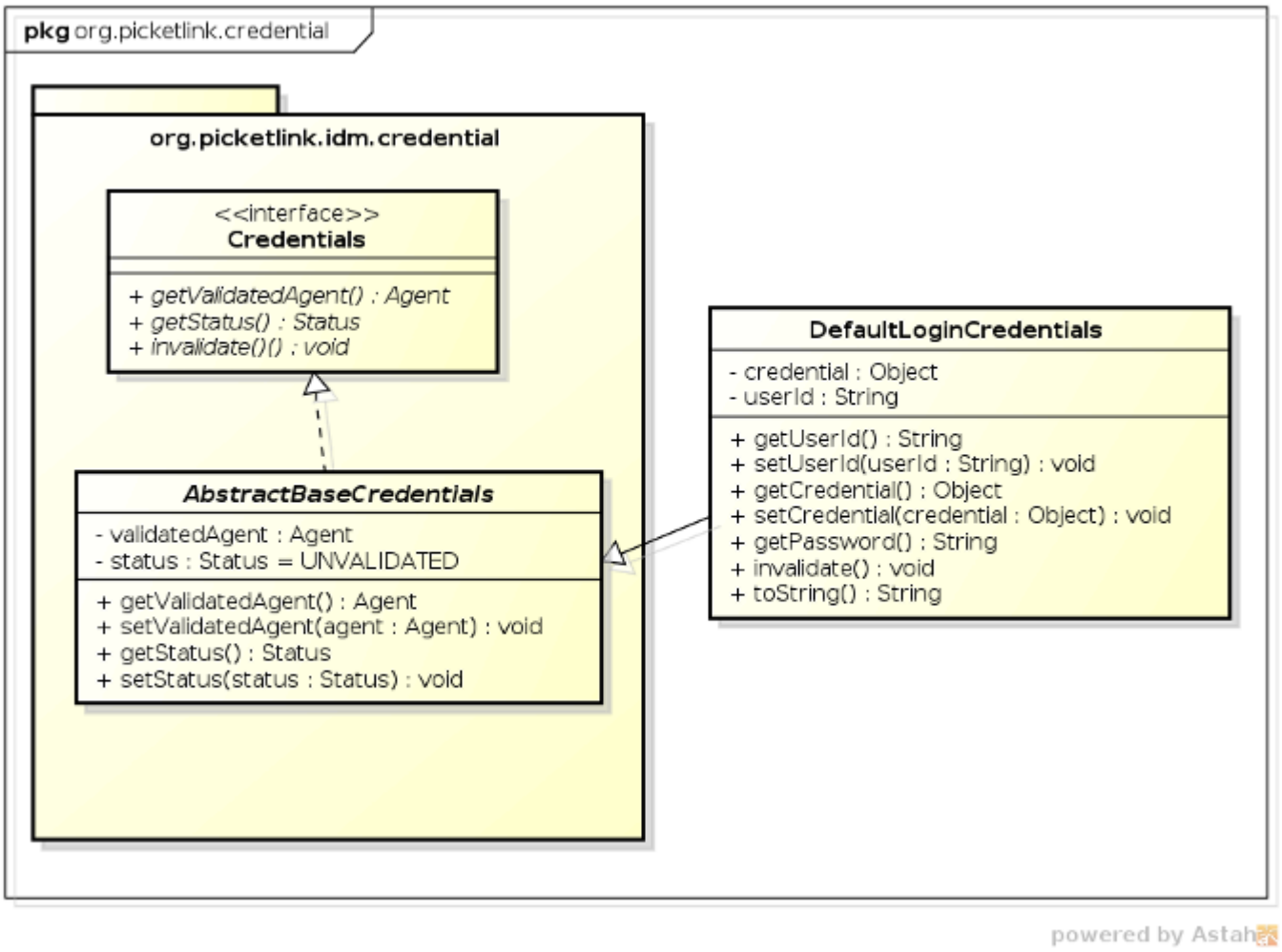
Of course it is not recommended that you actually do this, however this simplistic example serves adequately for demonstrating the case in point.

Report a bug[8]

2.3.4. DefaultLoginCredentials

The **`DefaultLoginCredentials`** bean is provided by PicketLink as a convenience, and is intended to serve as a general purpose **`Credentials`** implementation suitable for a variety of use cases. It supports the setting of a **`userId`** and **`credential`** property, and provides convenience methods for working with text-based passwords. It is a request-scoped bean and is also annotated with **`@Named`** so as to make it accessible directly from the view layer.

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29669-620275+%5BLatest%5D&comment=Title%3A+Credentials%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29669-620275+12+Mar+2014+12%3A30+en-US+%5BLatest%5D



A view technology with support for EL binding (such as JSF) can access the **`DefaultLoginCredentials`** bean directly via its bean name, **`loginCredentials`**. The following code snippet shows some JSF markup that binds the controls of a login form to **`DefaultLoginCredentials`**:

```
<div class="loginRow">
  <h:outputLabel for="name" value="Username" styleClass="loginLabel"/>
  <h:inputText id="name" value="#{loginCredentials.userId}"/>
</div>

<div class="loginRow">
  <h:outputLabel for="password" value="Password" styleClass="loginLabel"/>
  <h:inputSecret id="password" value="#{loginCredentials.password}" redisplay="true"/>
</div>
```

Report a bug[9]

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29670-675022+%5BLatest%5D&comment=Title%3A+DefaultLoginCredentials%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29670-675022+24+Jun+2014+00%3A14+en-US+%5BLatest%5D

2.4. Events

During the authentication and logout processes PicketLink will fire a few events to indicate that an action should be taken or to indicate that user's state has changed. They provide a nice and clean way to customize behavior, like providing user statistics or enable auditing to your application.

Table 2.1. Authentication Events

Event Type	Description
`org.picketlink.authentication.event.PreAuthenticationEvent`	This event is raised before authentication.
`org.picketlink.authentication.event.LoggedInEvent`	This event is raised when user successfully logs in.
`org.picketlink.authentication.event.PostAuthenticationEvent`	This event is raised after authentication.
`org.picketlink.authentication.event.AlreadyLoggedInEvent`	This event is raised when an already authenticated user attempts to authenticate again.
`org.picketlink.authentication.event.LoginFailedEvent`	This event is raised when an authentication attempt fails.
`org.picketlink.authentication.event.LockedAccountEvent`	This event is raised during the authentication process if the **`Account`** is disabled.
`org.picketlink.authentication.event.PreLoggedOutEvent`	This event is raised just before the user un-authenticates.
`org.picketlink.authentication.event.PostLoggedOutEvent`	This event is raised just after the user un-authenticates.

You can observe any of those events by providing a **`@Observer`** method to any bean, as follows:

```
public class AuthenticationAuditor {

 public void onSuccessFulLogin(@Observes LoggedInEvent event) {
  // handle user authenticated
 }

 public void onLoginFailed(@Observes LoginFailedEvent event) {
  // login failed
 }
}
```

Report a bug[10]

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30838-638598+%5BLatest%5D&comment=Title%3A+Events%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30838-638598+07+May+2014+01%3A25+en-US+%5BLatest%5D

Identity Management - Overview

3.1. Introduction

PicketLink Identity Management (IDM) is a fundamental module of PicketLink, with all other modules building on top of the IDM component to implement their extended features. It features provide a rich and extensible API for managing the identities (such as users, groups and roles) of your applications and services. It also supports a flexible system for identity partitioning, allowing it to be used as a complete security solution in simple web applications and/or as an Identity Provider (IDP) in more complex multi-domain scenarios. It also provides the core Identity Model API classes (see below) upon which an application's identity classes are defined to provide the security structure for that application.

The Identity Management features of PicketLink are accessed primarily via the following three interfaces:

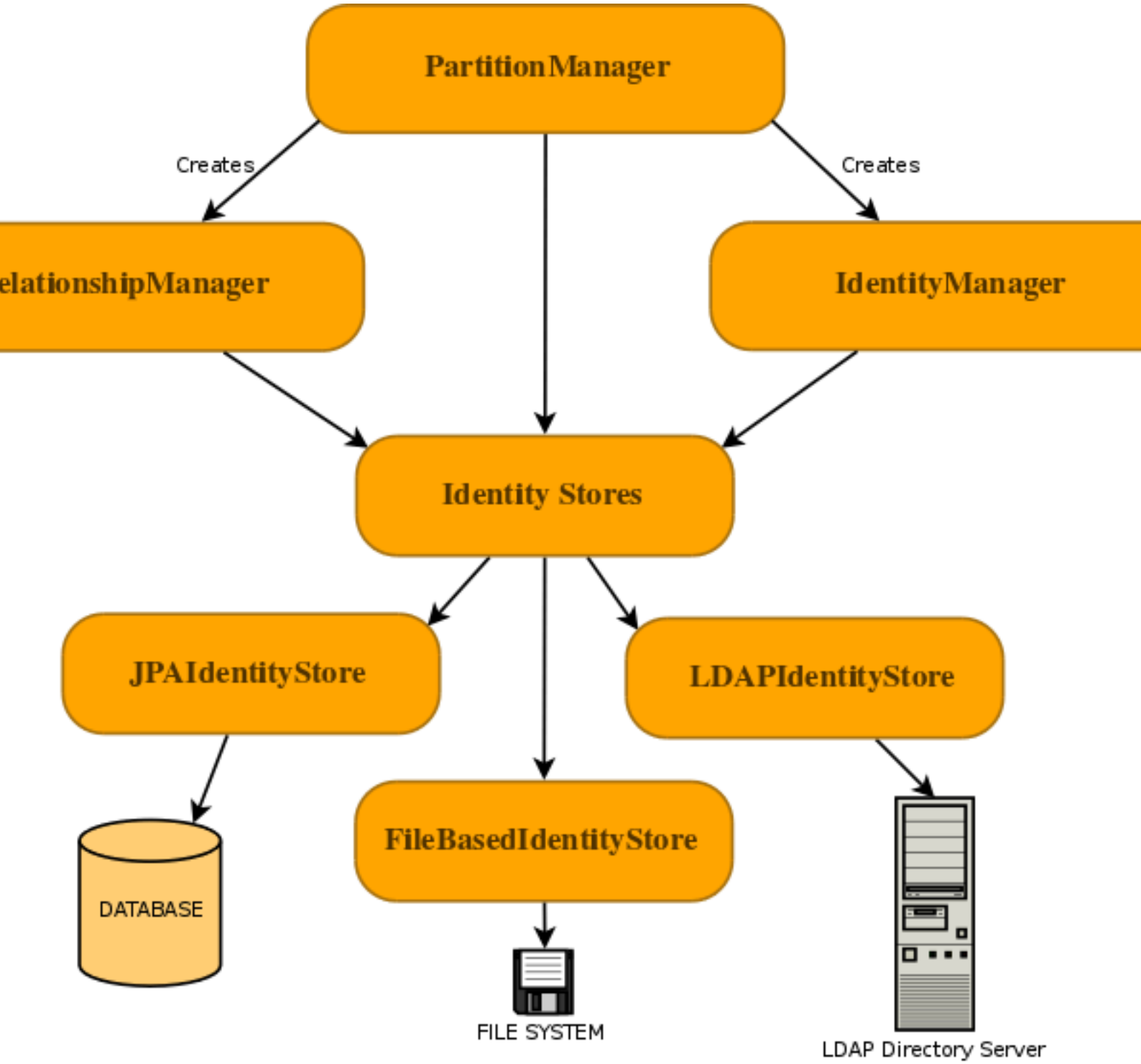
- **`PartitionManager`** is used to manage identity *partitions*, which are essentially a *container* for a set of identity objects. The **`PartitionManager`** interface provides a set of CRUD methods for creating, reading, updating and deleting partitions, as well as methods for creating an **`IdentityManager`** or **`RelationshipManager`** (more on these next). A typical Java EE application with simple security requirements will likely not be required to access the **`PartitionManager`** API directly.
- **`IdentityManager`** is used to manage *identity objects* within the scope of a partition. Some examples of identity objects are users, groups and roles, although PicketLink is not limited to just these. Besides providing the standard set of CRUD methods for managing and locating identity objects, the **`IdentityManager`** interface also defines methods for managing credentials and for creating identity queries which may be used to locate identities by their properties and attributes.
- **`RelationshipManager`** is used to manage *relationships* - a relationship is a typed association between two or more identities, with each identity having a definitive meaning within the relationship. Some examples of relationships that may already be familiar are group memberships (where a user is a member of a particular group) or granted roles (where a user is assigned to a role to afford them a certain set of privileges). The **`RelationshipManager`** provides CRUD methods for managing relationships, and also for creating a relationship query which may be used to locate relationships between identities based on the relationship type and participating identity object/s.
- **`PermissionManager`** is used to manage *permissions*. For more details take a look at *Chapter 10, Identity Management - Permissions API and Permission Management*.

Note

In case you are wondering why a separate **`RelationshipManager`** interface is required for managing relationships between identites, it is because PicketLink supports relationships between identities belonging to separate partitions; therefore the scope of a **`RelationshipManager`** instance is not constrained to a single partition in the same way as the **`IdentityManager`**.

Interaction with the backend store that provides the persistent identity state is performed by configuring one or more **`IdentityStore`**s. PicketLink provides a few built-in **`IdentityStore`**

implementations for storing identity state in a database, file system or LDAP directory server, and it is possible to provide your own custom implementation to support storing your application's identity data in other backends, or extend the built-in implementations to override their default behaviour.



Report a bug[1]

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29671-713902+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

3.1.1. Injecting the Identity Management Objects

In a Java EE environment PicketLink provides a set of producer methods for injecting the primary identity management objects into your CDI beans. The following table outlines which IDM classes may be injected, and the CDI scope of each of the beans.

Table 3.1. Identity Management Objects

Class Name	Scope
`org.picketlink.idm.PartitionManager`	**`@ApplicationScoped`**
`org.picketlink.idm.IdentityManager`	**`@RequestScoped`**
`org.picketlink.idm.RelationshipManager`	**`@RequestScoped`**
`org.picketlink.idm.PermissionManager`	**`@RequestScoped`**

Report a bug[2]

3.1.2. Interacting with PicketLink IDM During Application Startup

Since the **`IdentityManager`** and **`RelationshipManager`** beans are request scoped beans (as per the above table) it is not possible to inject them directly into a **`@Startup`** bean as there is no request scope available at application startup time. Instead, if you wish to use the IDM API within a **`@Startup`** bean in your Java EE application you may inject the **`PartitionManager`** (which is application-scoped) from which you can then get references to the **`IdentityManager`** and **`RelationshipManager`**:

```
@Singleton
@Startup
public class InitializeSecurity {
    @Inject private PartitionManager partitionManager;

    @PostConstruct
    public void create() {
        // Create an IdentityManager
        IdentityManager identityManager = partitionManager.createIdentityManager();

        User user = new User("shane");
        identityManager.add(user);

        Password password = new Password("password");
        identityManager.updateCredential(user, password);

        // Create a RelationshipManager
        RelationshipManager relationshipManager =
 partitionManager.createRelationshipManager();

        // Create some relationships
    }
}
```

Report a bug[3]

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29672-713901+%5BLatest%5D&comment=Title%3A+Injecting+the+Identity+Management+Objects%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29672-713901+30+Sep+2014+01%3A46+en-US+%5BLatest%5D

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29673-620279+

3.1.3. Configuring the Default Partition

Since PicketLink has multiple-partition support, it is important to be able to control the associated partition for the injected **`IdentityManager`**.

By default, PicketLink will inject an **`IdentityManager`** for the *default Realm* (i.e. the **`org.picketlink.idm.model.basic.Realm`** partition with a name of *default*). If your application has basic security requirements then this might well be adequate, however if you wish to override this default behaviour then simply provide a producer method annotated with the **`@PicketLink`** qualifier that returns the default partition for your application:

```
@ApplicationScoped
public class DefaultPartitionProducer {
    @Inject PartitionManager partitionManager;

    @Produces
    @PicketLink
    public Partition getDefaultPartition() {
        return partitionManager.getPartition(Realm.class, "warehouse.dispatch");
    }
}
```

Report a bug[4]

3.2. Getting Started - The 5 Minute Guide

If you'd like to get up and running with IDM quickly, the good news is that PicketLink will provide a default configuration that stores your identity data on the file system if no other configuration is available. This means that if you have the PicketLink libraries in your project, you can simply inject the **`PartitionManager`**, **`IdentityManager`** or **`RelationshipManager`** beans into your own application and start using them immediately:

```
@Inject PartitionManager partitionManager;
@Inject IdentityManager identityManager;
@Inject RelationshipManager relationshipManager;
```

Once you have injected an **`IdentityManager`** you can begin creating users, groups and roles for your application:

Note

The following code examples make use of the classes provided as part of the *basic identity model* - see *Chapter 5, Identity Management - Basic Identity Model* for more information.

%5BLatest%5D&comment=Title%3A+Interacting+with+PicketLink+IDM+During+Application+Startup%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29673-620279+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29674-680716+%5BLatest%5D&comment=Title%3A+Configuring+the+Default+Partition%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29674-680716+02+Jul+2014+06%3A47+en-US+%5BLatest%5D

```
User user = new User("jdoe");
user.setFirstName("Jane");
user.setLastName("Doe");
identityManager.add(user);

Group group = new Group("employees");
identityManager.add(group);

Role admin = new Role("admin");
identityManager.add(admin);
```

Use the **`RelationshipManager`** to create relationships, such as role assignments and group memberships:

```
// Grant the admin role to the user
relationshipManager.add(new Grant(user, admin));

// Add the user to the employees group
relationshipManager.add(new GroupMembership(user, group));
```

The static methods provided by the **`org.picketlink.idm.model.basic.BasicModel`** class are based on the basic identity model and may be used to lookup various identity objects, or test whether certain relationships exist. These methods accept either an **`IdentityManager`** or **`RelationshipManager`** object as a parameter.

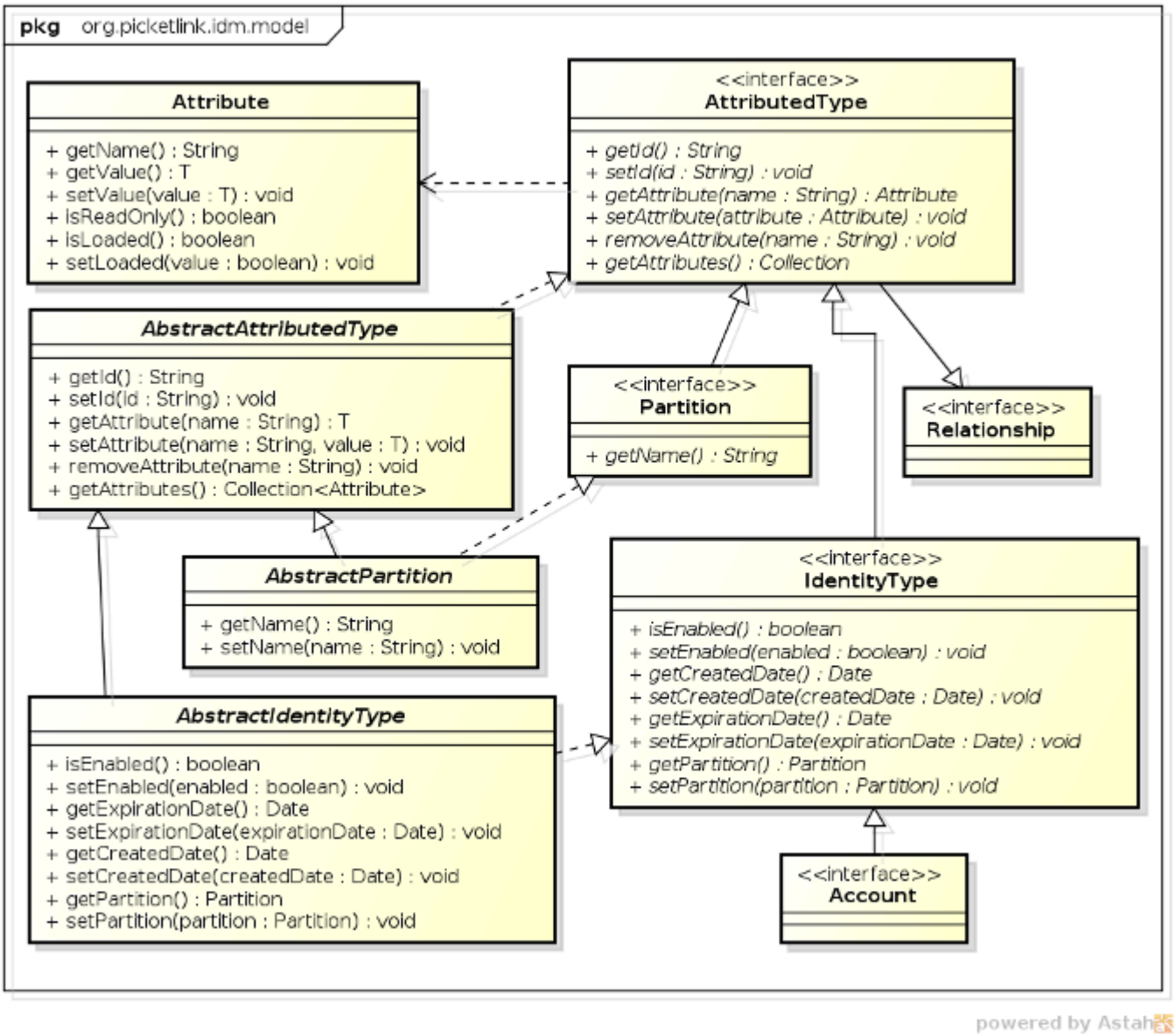
```
// Lookup the user by their username
User user = BasicModel.getUser(identityManager, "jdoe");

// Test if the user has the admin role
boolean isAdmin = BasicModel.hasRole(relationshipManager, user, role);

// Test if the user is a member of the employee group
boolean isEmployee = BasicModel.isMember(relationshipManager, user, group);
```

Report a bug[5]

3.3. Identity Model

The Identity Model is a set of classes that define the security structure of an application. It may consist of identity objects such as users, groups and roles; relationships such as group and role memberships; and partitions such as realms or tiers. The classes found in the **`org.picketlink.idm.model`** package define the base types upon which the identity model is built upon:

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29675-620655+%5BLatest%5D&comment=Title%3A+Getting+Started+-+The+5+Minute+Guide%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29675-620655+12+Mar+2014+12%3A35+en-US+%5BLatest%5D



- **`AttributedType`** is the base interface for the identity model. It declares a number of methods for managing a set of attribute values, plus **`getId()`** and **`setId()`** methods for setting a unique identifier value.
- **`Attribute`** is used to represent an attribute value. An attribute has a name and a (generically typed) value, and may be marked as read-only. Attribute values that are expensive to load (such as large binary data) may be lazy-loaded; the **`isLoaded()`** method may be used to determine whether the Attribute has been loaded or not.
- **`Partition`** is the base interface for partitions. Since each partition must have a name it declares a **`getName()`** method.
- **`Relationship`** is the base interface for relationships. Besides the base methods defined by the **`AttributedType`** interface, relationship implementations have no further contractual requirements, however they will define methods that return the identities and attribute values in accordance with the relationship type.

- **`IdentityType`** is the base interface for Identity objects. It declares properties that indicate whether the identity object is enabled or not, optional created and expiry dates, plus methods to read and set the owning **`Partition`**.
- **`Account`** is the base interface for identities that are capable of authenticating. Since the authentication process may not depend on one particular type of attribute (not all authentication is performed with a username and password) there are no hard-coded property accessors defined by this interface. It is up to each application to define the **`Account`** implementations required according to the application's requirements.
- **`AbstractAttributedType`** is an abstract base class for creating **`AttributedType`** implementations.
- **`AbstractPartition`** is an abstract base class that implements the base methods of the **`Partition`** interface, to simplify the development of partition implementations.
- **`AbstractIdentityType`** is an abstract base class that implements the base methods of the **`IdentityType`** interface, to simplify the development of identity objects.

Report a bug[6]

3.3.1. Which Identity Model Should My Application Use?

The base identity types listed above do not define an identity model implementation themselves, so they cannot be used directly to service the security requirements of an application. Instead, an application must either define its own identity model (by providing implementations of whichever identity objects are required by the application, such as user, group or role classes) or by using a pre-prepared model. PicketLink provides a *basic* identity model (more details can be found in *Chapter 5, Identity Management - Basic Identity Model*) which provides a basic set of identity objects, however in case the basic identity model is insufficient, it is quite simple to define a custom model as we'll see in the next section.

Report a bug[7]

3.4. Stereotypes

There is a set of recurring security concepts you will find in different applications such as:

- **`Users`**
- **`Roles`**
- **`Groups`**
- **`User's Roles Relationship`**
- **`Users and Group Membership`**

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29676-675024+%5BLatest%5D&comment=Title%3A+Identity+Model%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29677-620656+%5BLatest%5D&comment=Title%3A+Which+Identity+Model+Should+My+Application+Use%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29677-620656+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

Most applications use all or a set of these concepts in order to represent their users, roles, groups, how users are associated with roles and groups, etc. In PicketLink, those concepts are represented as *Stereotypes*.

As you might know, PicketLink provides a very extensible *Identity Model* from which you can extend in order to define your own representation for users, roles, groups, relationships between those entities and so forth. A *Stereotype* helps PicketLink to identify what a specific **`IdentityType`** or **`Relationship`** type is.

There are two types of stereotypes:

- **`Identity Stereotypes`**
- **`Relationship Stereotypes`**

A *Stereotype* is a very important concept in PicketLink Identity Management, specially when you need to customize it in order to provide your own and custom identity model. Some built-in features (eg.: credential validation, authorization, etc) are strongly based on those recurring concepts and helps PicketLink to fully integrate and recognize your custom identity model in order to reuse those features.

Important

Stereotypes don't imply you can't provide other representations for your own concepts. The Identity Model provided by PicketLink can helps you to represent anything you want. For example, in a typical mobile application you may want to represent user's devices as a specific identity type. So you can start applying security considering how user is accessing your application.

Report a bug[8]

3.4.1. Identity Stereotypes

Identity Stereotypes represent those recurring concepts applied to a specific **`IdentityType`** type. Let's take for an example the **`User`** type provided by the *Basic Model*:

```
@IdentityStereotype(USER)
public class User extends Agent {
}
```

Considering that the **`User`** type represents users we need to annotate the class with the **`@IdentityStereotype(USER)`** annotation.

This annotations supports the following stereotypes:

- **`USER`** - Should be used by identity types that represent an user.
- **`ROLE`** - Should be used by identity types that represent a role.

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A36178-666469+%5BLatest%5D&comment=Title%3A+Stereotypes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

- **GROUP** - Should be used by identity types that represent a group.

Each stereotype has its own common set of properties. Those properties represent some specific information which is usually associated with a stereotype. For example, roles usually have a **name** property from where we can get its name.

Stereotype properties are defined using the **StereotypeProperty** annotation. It provides a set of values defining the properties for a specific stereotype.

```
@IdentityStereotype(ROLE)
public class Role extends AbstractIdentityType {

    @AttributeProperty
    @StereotypeProperty(IDENTITY_ROLE_NAME)
    @Unique
    public String getName() {
        return this.name;
    }

}
```

The code example above defines that the **name** property of the **Role** type is related with the **name** of a role stereotype.

Report a bug[9]

3.4.2. Relationship Stereotypes

Relationship Stereotypes represent those recurring concepts applied to a specific **Relationship** type. Let's take for an example the **Grant** relationship type provided by the *Basic Model*:

```
@RelationshipStereotype(GRANT)
public class Grant extends AbstractAttributedType implements Relationship {
}
```

Considering that the **Grant** type represents the relationship between others identity types (eg.: users and groups) and roles, we need to annotate the class with the **@RelationshipStereotype(GRANT)** annotation.

This annotations supports the following stereotypes:

- **GRANT** - Should be used by relationship types that represent an association between any identity type and a role type.
- **GROUP_MEMBERSHIP** - Should be used by relationship types that represent an association between an identity type and a group type. Usually, the associated identity type is an user that is member of a group.

Each stereotype has its own common set of properties. Those properties represent some specific information which is usually associated with a stereotype. For example, grant stereotypes usually have a **assignee** and a **role** property.

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+36179-666465+%5BLatest%5D&comment=Title%3A+Identity+Stereotypes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=36179-666465+11+Jun+2014+02%3A50+en-US+%5BLatest%5D

Stereotype properties are defined using the **`StereotypeProperty`** annotation. It provides a set of values defining the properties for a specific stereotype.

```
@RelationshipStereotype(GRANT)
public class Grant extends AbstractAttributedType implements Relationship {

    @InheritsPrivileges("role")
    @StereotypeProperty(RELATIONSHIP_GRANT_ASSIGNEE)
    public IdentityType getAssignee() {
        return this.assignee;
    }

    @StereotypeProperty(RELATIONSHIP_GRANT_ROLE)
    public Role getRole() {
        return this.role;
    }

}
```

The code example above defines that the **`assignee`** property of the **`Role`** type is related with the **`assignee`** of a grant relationship stereotype. The same with the **`role`** property, which is related with the role assigned **`role`** to an assignee.

Report a bug[10]

3.5. Creating a Custom Identity Model

A custom identity model typically consists of two types of objects - the *identity* objects which define the security constructs of an application, and the *relationships* which define how the identity objects interract with each other to establish a meaningful security policy. PicketLink treats both types of object in an abstract manner, so it is up to the developer to create meaning for these objects and their relationships within the context of their own application. This section will describe how to create new identity objects and customize their properties and attributes, while the following section will complete the picture by describing how custom relationships are created.

Let's start by looking at the source for some of the identity objects in the basic model, starting with the **`Agent`** and **`User`** objects:

```
@IdentityStereotype(USER)
public class Agent extends AbstractIdentityType implements Account {
    private String loginName;

    public Agent() { }

    public Agent(String loginName) {
        this.loginName = loginName;
    }

    @AttributeProperty
    @StereotypeProperty(IDENTITY_USER_NAME)
    @Unique
    public String getLoginName() {
```

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+36180-666464+%5BLatest%5D&comment=Title%3A+Relationship+Stereotypes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=36180-666464+11+Jun+2014+02%3A50+en-US+%5BLatest%5D

```
        return loginName;
    }

    public void setLoginName(String loginName) {
        this.loginName = loginName;
    }
}
```

```
@IdentityStereotype(USER)
public class User extends Agent {
    private String firstName;
    private String lastName;
    private String email;

    public User() { }

    public User(String loginName) {
        super(loginName);
    }

    @AttributeProperty
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    @AttributeProperty
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    @AttributeProperty
    public String getEmail() {
        return this.email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

The **`Agent`** class is intended to represent a third party entity that may authenticate against an application, whether human (a user) or non-human (an external or remote process). Because **`Agent`** implements the **`Account`** marker interface, it is making a declaration that this identity object is capable of authenticating. To support the typical username/password authentication method, the **`Agent`** class defines a **`loginName`** property, however since the **`Account`** interface enforces no particular method of authentication (instead of a using username for authentication your application may require a certificate or fingerprint) this property is arbitrary.

The **`User`** class represents a human user and extends **`Agent`** to add the human-specific properties *firstName*, *lastName* and *email*. Since human users are also capable of authenticating it will also inherit the **`loginName`** property from the **`Agent`**.

Report a bug[11]

3.5.1. The `@AttributeProperty` Annotation

In the code above we can see that the getter methods of the identity objects are annotated with **`@AttributeProperty`**. This annotation (from the **`org.picketlink.idm.model.annotation`** package) is used to indicate that the property of the identity object should be persisted by the configured identity store when creating or updating the identity object. If this annotation was missing, then the property value would be **`null`** when loading the identity object from the identity store.

In this example, the annotation is placed on the getter method however it is also valid to place it on the corresponding field value.

Report a bug[12]

3.5.2. The `@Unique` Annotation

In the above code listing for the **`Agent`** class, we can also see that there is a **`@Unique`** annotation on the **`getLoginName()`** getter method (in addition to the **`@AttributeProperty`** annotation). This special annotation (also from the **`org.picketlink.idm.model.annotation`** package) is used to indicate to PicketLink that a unique constraint must be enforced on the property value - i.e. no two **`Agent`** objects (or their subclasses) may return the same value for **`getLoginName()`**.

Report a bug[13]

3.5.3. The `@InheritsPrivileges` Annotation

This annotation is used to configure privilege inheritence chains, and may either be applied to an **`IdentityType`** property of an **`IdentityType`** class, or to an **`IdentityType`** property of a **`Relationship`**.

When defined to an **`IdentityType`** property, it means instances inherit all privileges of the instance stored in the property annotated with **`@InheritsPrivileges`**. In the example below, childs groups inherit all privileges from their parents.

```
@IdentityStereotype(IdentityStereotype.Stereotype.GROUP)
public class Group extends AbstractIdentityType {

    @InheritsPrivileges
    @AttributeProperty
    public Group getParentGroup() {
```

[11] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29678-666472+%5BLatest%5D&comment=Title%3A+Creating+a+Custom+Identity+Model%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[12] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29679-620285+%5BLatest%5D&comment=Title%3A+The+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3E%40AttributeProperty%3C%2Fcode%3E+Annotation%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29679-620285+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

[13] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29680-620286+%5BLatest%5D&comment=Title%3A+The+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3E%40Unique%3C%2Fcode%3E+Annotation%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29680-620286+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

```
        return this.parentGroup;
    }

}
```

The same thing can be done for relationships, if you want to promote inheritance considering a hierarchy of an **`IdentityType`**.

```
@RelationshipStereotype(GRANT)
public class Grant extends AbstractAttributedType implements Relationship {

    @InheritsPrivileges("role")
    @StereotypeProperty(RELATIONSHIP_GRANT_ASSIGNEE)
    public IdentityType getAssignee() {
        return this.assignee;
    }

}
```

In this case, if this property is related with a **`Group`** instance, all childs of this group inherit the roles granted to it.

Report a bug[14]

3.6. Creating Custom Relationships

One of the strengths of PicketLink is its ability to support custom relationship types. This extensibility allows you, the developer to create specific relationship types between two or more identities to address the domain-specific requirements of your own application.

Note

Please note that custom relationship types are not supported by all **`IdentityStore`** implementations - see the Identity Store section above for more information.

To create a custom relationship type, we start by creating a new class that implements the **`Relationship`** interface. To save time, we also extend the **`AbstractAttributedType`** abstract class which takes care of the identifier and attribute management methods for us:

```
  public class Authorization extends AbstractAttributedType implements Relationship {

  }
```

The next step is to define which identities participate in the relationship. Once we create our identity property methods, we also need to annotate them with the

[14] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+38716-674395+%5BLatest%5D&comment=Title%3A+The+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3E%40InheritsPrivileges%3C%2Fcode%3E+Annotation%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=38716-674395+20+Jun+2014+06%3A10+en-US+%5BLatest%5D

`org.picketlink.idm.model.annotation.RelationshipIdentity` annotation. This is done by creating a property for each identity type.

```
  private User user;
  private Agent application;

  @RelationshipIdentity
  public User getUser() {
    return user;
  }

  public void setUser(User user) {
    this.user = user;
  }

  @RelationshipIdentity
  public Agent getApplication() {
    return application;
  }

  public void setApplication(Agent application) {
    this.application = application;
  }
```

We can also define some attribute properties, using the **`@RelationshipAttribute`** annotation:

```
  private String accessToken;

  @RelationshipAttribute
  public String getAccessToken() {
    return accessToken;
  }

  public void setAccessToken(String accessToken) {
    this.accessToken = accessToken;
  }
```

Report a bug[15]

3.7. Partition Management

PicketLink has been designed from the ground up to support a system of *partitioning*, allowing the identity objects it manages to be separated into logical groupings. Partitions may be used to split identities into separate *realms*, allowing an application to serve multiple organisations (for example in a SaaS architecture) or to support a multi-*tier* application allowing each tier to define its own set of identity objects (such as groups or roles). PicketLink's architecture also allows you to define your own custom partition types, allowing more complex use cases to be supported.

The **`PartitionManager`** interface provides the following methods for managing partitions:

```
public interface PartitionManager extends Serializable {

    <T extends Partition> T getPartition(Class<T> partitionClass, String name);
```

[15] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29681-620287+%5BLatest%5D&comment=Title%3A+Creating+Custom+Relationships%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29681-620287+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

```
    <T extends Partition> List<T> getPartitions(Class<T> partitionClass);

    <T extends Partition> T lookupById(final Class<T> partitionClass, String id);

    void add(Partition partition);

    void add(Partition partition, String configurationName);

    void update(Partition partition);

    void remove(Partition partition);
}
```

To create a new **`Partition`** object you may use either of the **`add()`** methods. If a **`configurationName`** parameter value isn't provided (see *Chapter 7, Identity Management - Configuration* for more information), then the newly created **`Partition`** will use the default configuration.

```
// Create a new Realm partition called "acme"
partitionManager.add(new Realm("acme"));
```

```
// Create a new Tier partition called "sales" using the named configuration "companyAD"
partitionManager.add(new Tier("sales"), "companyAD");
```

Each new **`Partition`** object created will be automatically assigned a unique identifier value, which can be accessed via its **`getId()`** method:

```
Realm realm = new Realm("acme");
partitionManager.add(realm);
String partitionId = realm.getId();
```

Partitions may be retrieved using either their name or their unique identifier value. Both methods require the exact partition class to be provided as a parameter:

```
Realm realm = partitionManager.getPartition(Realm.class, "acme");
Tier tier = partitionManager.lookupById(Tier.class, tierId);
```

It is also possible to retrieve all partitions for a given partition class. In this case you can retrieve all partitions for a given type or all of them:

```
List<Realm> realms = partitionManager.getPartitions(Realm.class);
List<Partition> allPartitions = partitionManager.getPartitions(Partition.class);
```

Since **`Partition`** objects all implement the **`AttributedType`** interface, it is also possible to set arbitrary attribute values:

```
realm.setAttribute(new Attribute<Date>("created", new Date()));
```

After making changes to an existing **`Partition`** object, the **`update()`** method may be used to persist those changes:

```
partitionManager.update(realm);
```

A **`Partition`** object may also be removed with the **`remove()`** method:

Warning

Removing a **`Partition`** object is permanent, and will also remove all identity objects that exist within that partition!

```
partitionManager.remove(realm);
```

Report a bug[16]

3.7.1. Creating Custom Partitions

Creating a custom partition type is extremely simple. PicketLink provides an abstract base class called **`AbstractPartition`** (see above) which makes creating a custom partition class a trivial exercise - simply extend the **`AbstractPartition`** class and then add any additional property getter/setter methods that you might require. Let's take a look at the built-in **`Realm`** class to see how little code it requires to create a custom partition:

```
@IdentityPartition(supportedTypes = {IdentityType.class})
public class Realm extends AbstractPartition {
    public Realm() {
        super(null);
    }

    public Realm(String name) {
        super(name);
    }
}
```

The **`@IdentityPartition`** annotation must be present on the partition class - the **`supportedTypes`** member is used to configure which identity types may be stored in this partition. Any identity object (or subclass) specified by **`supportedTypes`** is valid. There is also a **`unsupportedTypes`** member which may be used to specify identity types which *may not* be stored in the partition. This value can be used to *trim* unsupported classes (and their subclasses) off the **`supportedTypes`**.

Report a bug[17]

[16] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29682-620657+%5BLatest%5D&comment=Title%3A+Partition+Management%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[17] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29683-620289+%5BLatest%5D&comment=Title%3A+Creating+Custom+Partitions%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29683-620289+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

Identity Management - Credential Validation and Management

4.1. Authentication

Note

While the IDM module of PicketLink provides authentication features, for common use cases involving standard username and password based authentication in a Java EE environment, PicketLink provides a more streamlined method of authentication. Please refer to *Chapter 2, Authentication* for more information.

PicketLink IDM provides an authentication subsystem that allows user credentials to be validated thereby confirming that an authenticating user is who they claim to be. The **`IdentityManager`** interface provides a single method for performing credential validation, as follows:

```
void validateCredentials(Credentials credentials);
```

The **`validateCredentials()`** method accepts a single **`Credentials`** parameter, which should contain all of the state required to determine who is attempting to authenticate, and the credential (such as a password, certificate, etc) that they are authenticating with. Let's take a look at the **`Credentials`** interface:

```
public interface Credentials {
    public enum Status {
        UNVALIDATED, IN_PROGRESS, INVALID, VALID, EXPIRED
    };

   Account getValidatedAccount();

   Status getStatus();

   void invalidate();
}
```

- The **`Status`** enum defines the following values, which reflect the various credential states:
  - **`UNVALIDATED`** - The credential is yet to be validated.
  - **`IN_PROGRESS`** - The credential is in the process of being validated.
  - **`INVALID`** - The credential has been validated unsuccessfully
  - **`VALID`** - The credential has been validated successfully
  - **`EXPIRED`** - The credential has expired
- **`getValidatedAccount()`** - If the credential was successfully validated, this method returns the **`Account`** object representing the validated user.

- **`getStatus()`** - Returns the current status of the credential, i.e. one of the above enum values.
- **`invalidate()`** - Invalidate the credential. Implementations of **`Credential`** should use this method to clean up internal credential state.

Let's take a look at a concrete example - **`UsernamePasswordCredentials`** is a **`Credentials`** implementation that supports traditional username/password-based authentication:

```
public class UsernamePasswordCredentials extends AbstractBaseCredentials {

    private String username;

    private Password password;

    public UsernamePasswordCredentials() { }

    public UsernamePasswordCredentials(String userName, Password password) {
        this.username = userName;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public UsernamePasswordCredentials setUsername(String username) {
        this.username = username;
        return this;
    }

    public Password getPassword() {
        return password;
    }

    public UsernamePasswordCredentials setPassword(Password password) {
        this.password = password;
        return this;
    }

    @Override
    public void invalidate() {
        setStatus(Status.INVALID);
        password.clear();
    }
}
```

The first thing we may notice about the above code is that the **`UsernamePasswordCredentials`** class extends **`AbstractBaseCredentials`**. This abstract base class implements the basic functionality required by the **`Credentials`** interface. Next, we can see that two fields are defined; **`username`** and **`password`**. These fields are used to hold the username and password state, and can be set either via the constructor, or by their associated setter methods. Finally, we can also see that the **`invalidate()`** method sets the status to **`INVALID`**, and also clears the password value.

Let's take a look at an example of the above classes in action. The following code demonstrates how we would authenticate a user with a username of "john" and a password of "abcde":

```
Credentials creds = new UsernamePasswordCredentials("john",
    new Password("abcde"));
identityManager.validate(creds);
if (Status.VALID.equals(creds.getStatus())) {
  // authentication was successful
}
```

We can also test if the credentials that were provided have expired (if an expiry date was set). In this case we might redirect the user to a form where they can enter a new password.

```
Credentials creds = new UsernamePasswordCredentials("john",
    new Password("abcde"));
identityManager.validate(creds);
if (Status.EXPIRED.equals(creds.getStatus())) {
  // password has expired, redirect the user to a password change screen
}
```

Report a bug[1]

4.2. Managing Credentials

Updating user credentials is even easier than validating them. The **`IdentityManager`** interface provides the following two methods for updating credentials:

```
void updateCredential(Account account, Object credential);
void updateCredential(Account account, Object credential, Date effectiveDate, Date
 expiryDate);
```

Both of these methods essentially do the same thing; they update a credential value for a specified **`Account`**. The second overloaded method however also accepts **`effectiveDate`** and **`expiryDate`** parameters, which allow some temporal control over when the credential will be valid. Use cases for this feature include implementing a strict password expiry policy (by providing an expiry date), or creating a new account that might not become active until a date in the future (by providing an effective date). Invoking the first overloaded method will store the credential with an effective date of the current date and time, and no expiry date.

Note

One important point to note is that the **`credential`** parameter is of type **`java.lang.Object`**. Since credentials can come in all shapes and sizes (and may even be defined by third party libraries), there is no common base interface for credential implementations to extend. To support this type of flexibility in an extensible way, PicketLink provides an SPI that allows custom credential handlers to be configured that override or extend the default credential handling logic. Please see the next section for more information on how this SPI may be used.

Let's take a look at a couple of examples. Here's some code demonstrating how a password can be assigned to user "jsmith":

```
User user = BasicModel.getUser(identityManager, "jsmith");
identityManager.updateCredential(user, new Password("abcd1234"));
```

This example creates a digest and assigns it to user "jdoe":

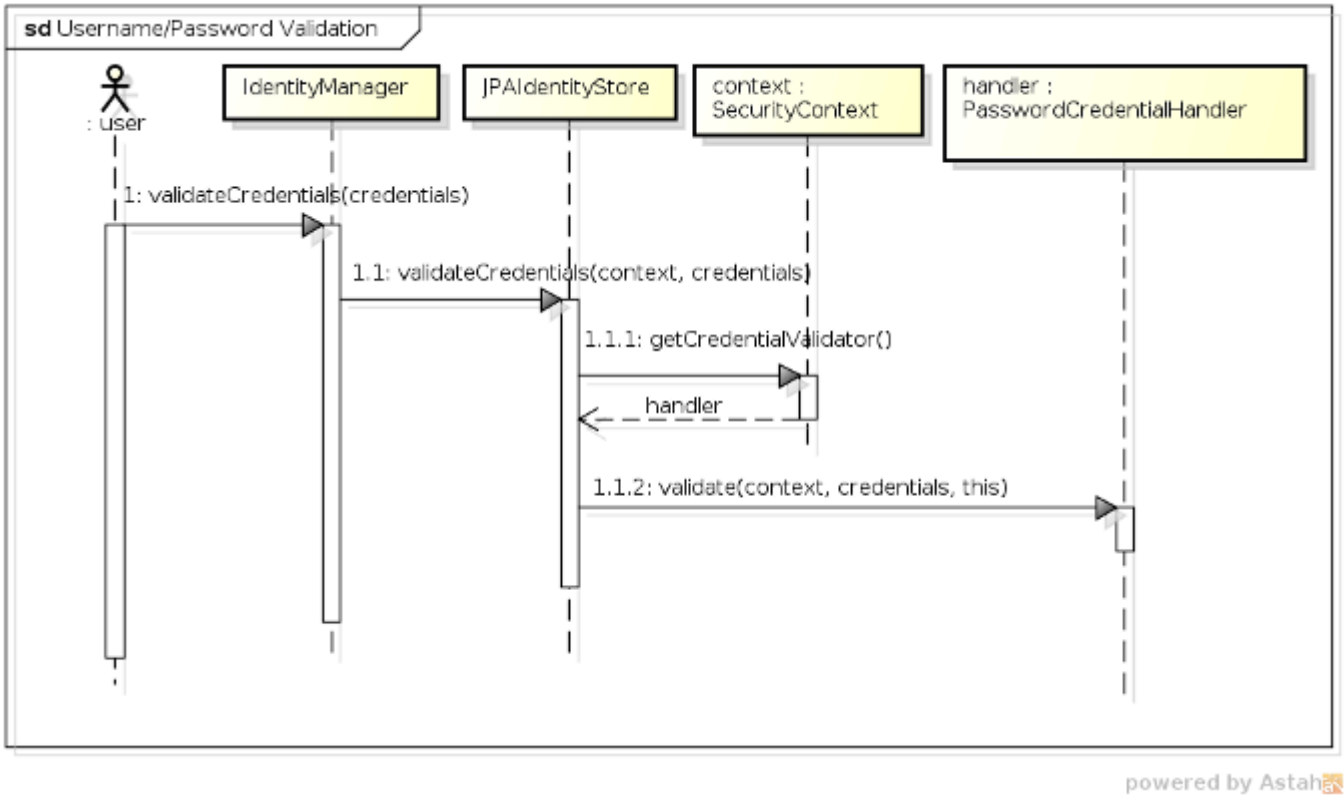
[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29684-620993+%5BLatest%5D&comment=Title%3A+Authentication%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29684-620993+13+Mar+2014+09%3A47+en-US+%5BLatest%5D

```
User user = BasicModel.getUser(identityManager, "jdoe");
Digest digest = new Digest();
digest.setRealm("default");
digest.setUsername(user.getLoginName());
digest.setPassword("abcd1234");
identityManager.updateCredential(user, digest);
```

Report a bug[2]

4.3. Credential Handlers

For **`IdentityStore`** implementations that support multiple credential types, PicketLink provides an optional SPI to allow the default credential handling logic to be easily customized and extended. To get a better picture of the overall workings of the Credential Handler SPI, let's take a look at the sequence of events during the credential validation process when validating a username and password against **`JPAIdentityStore`**:



- *1* - The user (or some other code) first invokes the **`validateCredentials()`** method on **`IdentityManager`**, passing in the **`Credentials`** instance to validate.
- *1.1* - After looking up the correct **`IdentityStore`** (i.e. the one that has been configured to validate credentials) the **`IdentityManager`** invokes the store's **`validateCredentials()`** method, passing in the **`IdentityContext`** and the credentials to validate.

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29685-620291+%5BLatest%5D&comment=Title%3A+Managing+Credentials%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29685-620291+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

- *1.1.1* - In **`JPAIdentityStore`**'s implementation of the **`validateCredentials()`** method, the **`IdentityContext`** is used to look up the **`CredentialHandler`** implementation that has been configured to process validation requests for usernames and passwords, which is then stored in a local variable called **`handler`**.
- *1.1.2* - The **`validate()`** method is invoked on the **`CredentialHandler`**, passing in the security context, the credentials value and a reference back to the identity store. The reference to the identity store is important as the credential handler may require it to invoke certain methods upon the store to validate the credentials.

The **`CredentialHandler`** interface declares three methods, as follows:

```
public interface CredentialHandler {
    void setup(IdentityStore<?> identityStore);

    void validate(IdentityContext context, Credentials credentials,
                  IdentityStore<?> identityStore);

    void update(IdentityContext context, Account account, Object credential,
                IdentityStore<?> identityStore, Date effectiveDate, Date expiryDate);

}
```

The **`setup()`** method is called once, when the **`CredentialHandler`** instance is first created. Credential handler instantiation is controlled by the **`CredentialHandlerFactory`**, which creates a single instance of each **`CredentialHandler`** implementation to service all credential requests for that handler. Each **`CredentialHandler`** implementation must declare the types of credentials that it is capable of supporting, which is done by annotating the implementation class with the **`@SupportsCredentials`** annotation like so:

```
@SupportsCredentials(
  credentialClass = { UsernamePasswordCredentials.class, Password.class },
  credentialStorage = EncodedPasswordStorage.class
)
public class PasswordCredentialHandler implements CredentialHandler {
```

Since the **`validate()`** and **`update()`** methods receive different parameter types (**`validate()`** takes a **`Credentials`** parameter value while **`update()`** takes an **`Object`** that represents a single credential value), the **`@SupportsCredentials`** annotation must contain a complete list of all types supported by that handler.

Similarly, if the **`IdentityStore`** implementation makes use of the credential handler SPI then it also must declare which credential handlers support that identity store. This is done using the **`@CredentialHandlers`** annotation; for example, the following code shows how **`JPAIdentityStore`** is configured to be capable of handling credential requests for usernames and passwords, X509 certificates and digest-based authentication:

```
@CredentialHandlers({ PasswordCredentialHandler.class,
          X509CertificateCredentialHandler.class, DigestCredentialHandler.class })
public class JPAIdentityStore implements IdentityStore<JPAIdentityStoreConfiguration>,
                                         CredentialStore {
```

Report a bug[3]

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29686-675025+%5BLatest

4.3.1. The `CredentialStore` interface

For **`IdentityStore`** implementations that support multiple credential types (such as **`JPAIdentityStore`** and **`FileBasedIdentityStore`**), the implementation may choose to also implement the **`CredentialStore`** interface to simplify the interaction between the **`CredentialHandler`** and the **`IdentityStore`**. The **`CredentialStore`** interface declares methods for storing and retrieving credential values within an identity store, as follows:

```
public interface CredentialStore {
   /**
    * Stores the specified credential state.
    *
    * @param context The contextual invocation context.
    * @param account The account which credentials should be removed.
    * @param storage The credential storage instance to be stored.
    */
   void storeCredential(IdentityContext context, Account account, CredentialStorage
storage);

   /**
    * Returns the currently active credential state of the specified {@link T}, for the
specified {@link org.picketlink.idm.model.Account}.
    *
    * @param context The contextual invocation context.
    * @param account The account which credentials should be removed.
    * @param storageClass The credential storage type specifying which credential types
should be removed.
    *
    * @return
    */
   <T extends CredentialStorage> T retrieveCurrentCredential(IdentityContext context,
Account account, Class<T> storageClass);

   /**
    * Returns a list of all credential state of the specified {@link T}, for the
specified {@link org.picketlink.idm.model.Account}.
    *
    * @param context The contextual invocation context.
    * @param account The account which credentials should be removed.
    * @param storageClass The credential storage type specifying which credential types
should be removed.
    *
    * @return
    */
   <T extends CredentialStorage> List<T> retrieveCredentials(IdentityContext context,
Account account, Class<T> storageClass);

   /**
    * <p>Removes all credentials stored by a certain {@link
org.picketlink.idm.credential.storage.CredentialStorage} associated
    * with the given {@link org.picketlink.idm.model.Account}.</p>
    *
    * @param context The contextual invocation context.
    * @param account The account which credentials should be removed.
    * @param storageClass The credential storage type specifying which credential types
should be removed.
    */
   void removeCredential(IdentityContext context, Account account, Class<? extends
CredentialStorage> storageClass);
}
```

%5D&comment=Title%3A+Credential+Handlers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

Report a bug[4]

4.3.2. The CredentialStorage interface

The **CredentialStorage** interface is essentially used to represent the state required to validate an account's credentials, and is persisted within the identity store. The base interface is quite simple and only declares two methods - **getEffectiveDate()** and **getExpiryDate()**:

```
public interface CredentialStorage {
   @Stored Date getEffectiveDate();
   @Stored Date getExpiryDate();
}
```

The most significant thing to note above is the usage of the **@Stored** annotation. This annotation is used to mark the properties of the **CredentialStorage** implementation that should be persisted. The only requirement for any property values that are marked as **@Stored** is that they are serializable (i.e. they implement the **java.io.Serializable** interface). The **@Stored** annotation may be placed on either the getter method or the field variable itself. An implementation of **CredentialStorage** will typically declare a number of properties (in addition to the **effectiveDate** and **expiryDate** properties) annotated with **@Stored**. Here's an example of one of a **CredentialStorage** implementation that is built into PicketLink - **EncodedPasswordStorage** is used to store a password hash and salt value:

```
public class EncodedPasswordStorage implements CredentialStorage {

    private Date effectiveDate;
    private Date expiryDate;
    private String encodedHash;
    private String salt;

    @Override @Stored
    public Date getEffectiveDate() {
        return effectiveDate;
    }

    public void setEffectiveDate(Date effectiveDate) {
        this.effectiveDate = effectiveDate;
    }

    @Override @Stored
    public Date getExpiryDate() {
        return expiryDate;
    }

    public void setExpiryDate(Date expiryDate) {
        this.expiryDate = expiryDate;
    }

    @Stored
    public String getEncodedHash() {
        return encodedHash;
    }
```

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29687-686427+%5BLatest%5D&comment=Title%3A+The+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3ECredentialStore%3C%2Fcode%3E+interface%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29687-686427+22+Jul+2014+05%3A12+en-US+%5BLatest%5D

```
    public void setEncodedHash(String encodedHash) {
        this.encodedHash = encodedHash;
    }

    @Stored
    public String getSalt() {
        return this.salt;
    }

    public void setSalt(String salt) {
        this.salt = salt;
    }

}
```

Report a bug[5]

4.4. Built-in Credential Handlers

PicketLink provides built-in support for the following credential types:

Warning

Not all built-in **`IdentityStore`** implementations support all credential types. For example, since the **`LDAPIdentityStore`** is backed by an LDAP directory server, only password credentials are supported. The following table lists the built-in **`IdentityStore`** implementations that support each credential type.

Table 4.1. Built-in credential types

Credential type	Description	Supported by
`org.picketlink.idm.credential.Password`	Username and password-based password	**`JPAIdentityStore`** **`FileBasedIdentityStore`** **`LDAPIdentityStore`**
`org.picketlink.idm.credential.Digest`	Used for digest-based authentication	**`JPAIdentityStore`** **`FileBasedIdentityStore`**
`java.security.cert.X509Certificate`	Used for X509 certificate based authentication	**`JPAIdentityStore`** **`FileBasedIdentityStore`**
`org.picketlink.idm.credential.TOTPCredential`	Used for TOTP-based One-time Password authentication	**`JPAIdentityStore`** **`FileBasedIdentityStore`**

The next sections will describe each of these built-in types individually. Configuration parameters are set at initialization time - see *Section 7.1.8.1, "Passing parameters to Credential Handlers"* for details.

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29688-620294+%5BLatest%5D&comment=Title%3A+The+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3ECredentialStorage%3C%2Fcode%3E+interface%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29688-620294+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

Report a bug[6]

4.4.1. Username/Password-based Credential Handler

This credential handlers supports a username/password based authentication.

Credentials can be updated as follows:

```
User user = BasicModel.getUser(identityManager, "jsmith");
identityManager.updateCredential(user, new Password("abcd1234"));
```

In order to validate a credential you need to the following code:

```
UsernamePasswordCredentials credential = new UsernamePasswordCredentials();

Password password = new Password("abcd1234");

credential.setUsername("jsmith");
credential.setPassword(password);

identityManager.validateCredentials(credential);

if (Status.VALID.equals(credential.getStatus()) {
  // successful validation
} else {
  // invalid credential
}
```

Report a bug[7]

4.4.1.1. Configuration Parameters

The following table describes all configuration parameters supported by this credential handler:

Table 4.2. Configuration Parameters

Parameter	Description
PasswordCredentialHandler.PASSWORD_ENCODER	It must be a **org.picketlink.idm.credential.encoder.Password** sub-type. It defines how passwords are encoded. Defaults to SHA-512.
PasswordCredentialHandler.SECURE_RANDOM_PROVIDER	It must be a **org.picketlink.common.random.SecureRandomProvi** sub-type. It defines how **SecureRandom** are created in order to be used to generate random numbers to salt passwords. Defaults to SHA1PRNG with a default seed.

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29689-620659+%5BLatest%5D&comment=Title%3A+Built-in+Credential+Handlers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29690-620296+%5BLatest%5D&comment=Title%3A+Username%2FPassword-based+Credential+Handler%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

Parameter	Description
PasswordCredentialHandler.RENEW_RANDOM_NUMBER_GENERATOR_INTERVAL	To increase the security of generated salted passwords, **SecureRandom** instances can be renewed from time to time. This option defines the time in milliseconds. Defaults to disabled, what means that a single instance is used during the life-time of the application.
PasswordCredentialHandler.ALGORITHM_RANDOM_NUMBER	Defines the algorithm to be used by the default **SecureRandomProvider**. Defaults to SHA1PRNG.
PasswordCredentialHandler.KEY_LENGTH_RANDOM_NUMBER	Defines the key length of seeds when using the default **SecureRandomProvider**. Defaults to 0, which means it is disabled.
PasswordCredentialHandler.LOGIN_NAME_PROPERTY	This option defines the name of the property used to lookup the **Account** object using the provided login name. It has a default value of **loginName** and can be overridden if the credential handler is to be used to authenticate an **Account** type that uses a different property name.
PasswordCredentialHandler.SUPPORTED_ACCOUNT_TYPES	This option defines any additional **Account** types that are supported by the credential handler. If no value is specified and/or no identity instances of the specified types are found then the credential handler's fall back behaviour is to attempt to lookup either an **Agent** or **User** (from the **org.picketlink.idm.model.basic** package) identity. The property value is expected to be an array of **Class<? extends Account>** objects.

Report a bug[8]

4.4.2. DIGEST-based Credential Handler

This credential handlers supports a DIGEST based authentication.

Credentials can be updated as follows:

```
User user = BasicModel.getUser(identityManager, "jsmith");
Digest digest = new Digest();

digest.setRealm("PicketLink Realm");
digest.setUsername(user.getLoginName());
digest.setPassword("abcd1234");
```

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29691-620660+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29691-620660+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

```
identityManager.updateCredential(user, digest);
```

In order to validate a credential you need to the following code:

```
User user = BasicModel.getUser(identityManager, "jsmith");

Digest digest = new Digest();

digest.setRealm("PicketLink Realm");
digest.setUsername(user.getLoginName());
digest.setPassword("abcd1234");

digest.setDigest(DigestUtil.calculateA1(user.getLoginName(), digest.getRealm(),
 digest.getPassword().toCharArray()));

DigestCredentials credential = new DigestCredentials(digest);

identityManager.validateCredentials(credential);

if (Status.VALID.equals(credential.getStatus()) {
// successful validation
} else {
// invalid credential
}
```

Report a bug[9]

4.4.3. X509-based Credential Handler

This credential handlers supports a X509 certificates based authentication.

Credentials can be updated as follows:

```
User user = BasicModel.getUser(identityManager, "jsmith");

java.security.cert.X509Certificate clientCert = // get user certificate

identityManager.updateCredential(user, clientCert);
```

In order to validate a credential you need to the following code:

```
User user = BasicModel.getUser(identityManager, "jsmith");

java.security.cert.X509Certificate clientCert = // get user certificate
X509CertificateCredentials credential = new X509CertificateCredentials(clientCert);

identityManager.validateCredentials(credential);

if (Status.VALID.equals(credential.getStatus()) {
// successful validation
} else {
// invalid credential
}
```

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29692-680715+%5BLatest%5D&comment=Title%3A+DIGEST-based+Credential+Handler%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29692-680715+02+Jul+2014+06%3A44+en-US+%5BLatest%5D

In some cases, you just want to trust the provided certificate and only check the existence of the principal:

```
User user = BasicModel.getUser(identityManager, "jsmith");

java.security.cert.X509Certificate clientCert = // get user certificate
X509CertificateCredentials credential = new X509CertificateCredentials(clientCert);

// trust the certificate and only check the principal existence
credential.setTrusted(true);

identityManager.validateCredentials(credential);

if (Status.VALID.equals(credential.getStatus()) {
// successful validation
} else {
// invalid credential
}
```

Report a bug[10]

4.4.4. Time-based One Time Password Credential Handler

This credential handlers supports a username/password based authentication.

Credentials can be updated as follows:

```
User user = BasicModel.getUser(identityManager, "jsmith");

TOTPCredential credential = new TOTPCredential("abcd1234", "my_totp_secret");

identityManager.updateCredential(user, credential);
```

Users can have multiple TOTP tokens, one for each device. You can provide configure tokens for a specific user device as follows:

```
User user = BasicModel.getUser(identityManager, "jsmith");

TOTPCredential credential = new TOTPCredential("abcd1234", "my_totp_secret");

credential.setDevice("My Cool Android Phone");

identityManager.updateCredential(user, credential);
```

In order to validate a credential you need to the following code:

```
User user = BasicModel.getUser(identityManager, "jsmith");

TOTPCredentials credential = new TOTPCredentials();

credential.setUsername(user.getLoginName());
credential.setPassword(new Password("abcd1234"));
```

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29693-620299+%5BLatest%5D&comment=Title%3A+X509-based+Credential+Handler%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29693-620299+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

```
TimeBasedOTP totp = new TimeBasedOTP();

// let's manually generate a token based on the user secret
String token = totp.generate("my_totp_secret");

credential.setToken(token);

// if you want to validate the token for a specific device
// credential.setDevice("My Cool Android Phone");

identityManager.validateCredentials(credential);

if (Status.VALID.equals(credential.getStatus()) {
  // successful validation
} else {
  // invalid credential
}
```

Report a bug[11]

4.4.4.1. Configuration Parameters

The following table describes all configuration parameters supported by this credential handler:

Table 4.3. Configuration Parameters

Parameter	Description
`TOTPCredentialHandler.ALGORITHM`	The encryption algorithm. Defaults to HmacSHA1.
`TOTPCredentialHandler.INTERVAL_SECONDS`	The number of seconds a token is valid. Defaults to 30 seconds.
`TOTPCredentialHandler.NUMBER_DIGITS`	The number of digits for a token. Defaults to 6 digits.
`TOTPCredentialHandler.DELAY_WINDOW`	the number of previous intervals that should be used to validate tokens. Defaults to 1 interval of 30 seconds.

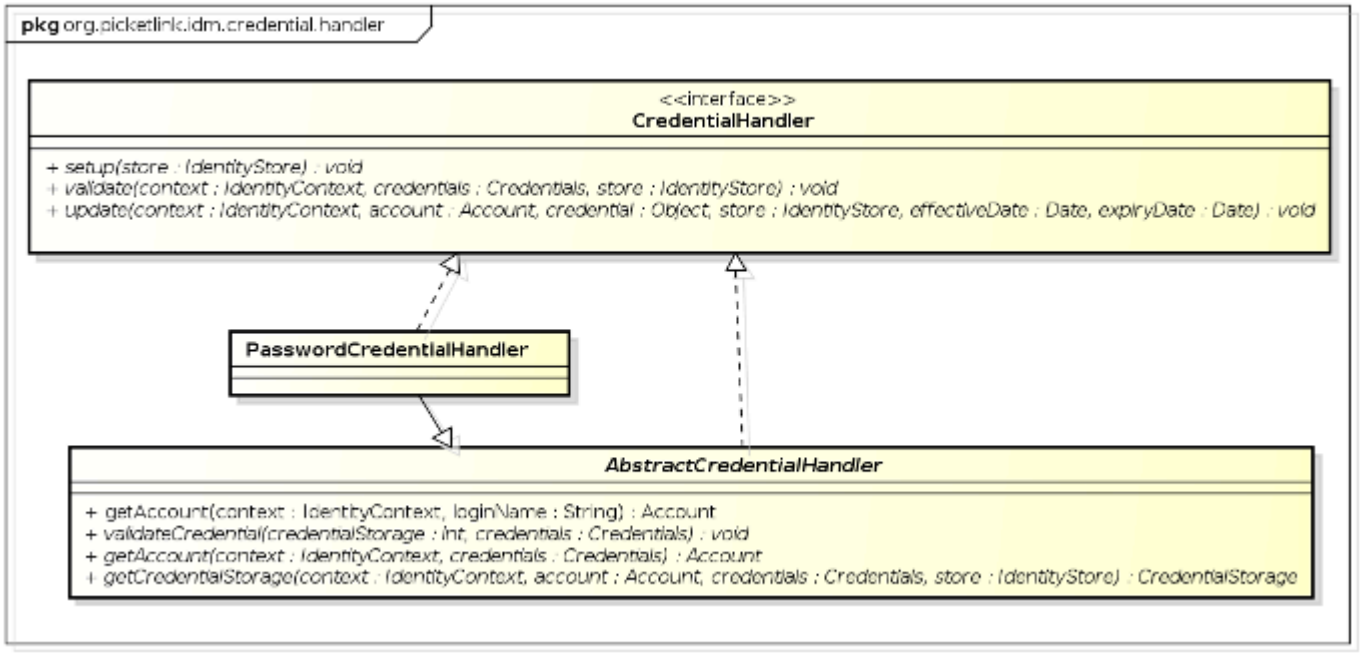
Report a bug[12]

4.5. Implementing a Custom `CredentialHandler`

In this section we'll dissect the **`PasswordCredentialHandler`** to learn how to create a custom credential handler. The **`AbstractCredentialHandler`** abstract class is provided to simplify the process of creating a new credential handler, and is also used by **`PasswordCredentialHandler`** as a base class:

[11] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29694-620300+%5BLatest%5D&comment=Title%3A+Time-based+One+Time+Password+Credential+Handler%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[12] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29695-620661+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29695-620661+12+Mar+2014+12%3A35+en-US+%5BLatest%5D



Let's start by looking at the class declaration for **`PasswordCredentialHandler`**:

```
@SupportsCredentials(
        credentialClass = {UsernamePasswordCredentials.class, Password.class},
        credentialStorage = EncodedPasswordStorage.class)
public class PasswordCredentialHandler<S extends CredentialStore<?>,
    V extends UsernamePasswordCredentials,
    U extends Password>
    extends AbstractCredentialHandler<S, V, U> {
```

The **`@SupportsCredentials`** annotation is used to declare exactly which credential classes are supported by the credential handler (as indicated by the **`credentialClass`** annotation member). The supported credential classes include both credentials used to authenticate via the **`validate()`** method (i.e. a class that implements the **`org.picketlink.idm.credential.Credentials`** interface, in this example **`UsernamePasswordCredentials`**) and the actual encapsulated credential value (e.g. **`org.picketlink.idm.credential.Password`**). These supported credential classes are also reflected in the generic type declaration of the class itself - the **`V`** and **`U`** types in the code above. The **`credentialStorage`** annotation member declares the storage class used to persist the necessary state for the credential. The storage class must implement the **`org.picketlink.idm.credential.storage.CredentialStorage`** interface.

The **`setup()`** method is executed only once and is used to perform any required initialization for the credential handler. In the case of **`PasswordCredentialHandler`**, the **`setup()`** method reads the configuration properties (made available from **`store.getConfig().getCredentialHandlerProperties()`**) and uses those property values to initialize the state required for encoding password values.

```
@Override
public void setup(S store) {
    super.setup(store);

    Map<String, Object> options = store.getConfig().getCredentialHandlerProperties();

    if (options != null) {
        Object providedEncoder = options.get(PASSWORD_ENCODER);

        if (providedEncoder != null) {
            if (PasswordEncoder.class.isInstance(providedEncoder)) {
```

```
                this.passwordEncoder = (PasswordEncoder) providedEncoder;
            } else {
                throw new SecurityConfigurationException("The password encoder [" +
                    providedEncoder + "] must be an instance of " +
                    PasswordEncoder.class.getName());
            }
        }

        Object renewRandomNumberGeneratorInterval = options.get(
            RENEW_RANDOM_NUMBER_GENERATOR_INTERVAL);

        if (renewRandomNumberGeneratorInterval != null) {
            this.renewRandomNumberGeneratorInterval = Integer.valueOf(
                renewRandomNumberGeneratorInterval.toString());
        }

        Object secureRandomProvider = options.get(SECURE_RANDOM_PROVIDER);

        if (secureRandomProvider != null) {
            this.secureRandomProvider = (SecureRandomProvider) secureRandomProvider;
        } else {
            Object saltAlgorithm = options.get(ALGORITHM_RANDOM_NUMBER);

            if (saltAlgorithm == null) {
                saltAlgorithm = DEFAULT_SALT_ALGORITHM;
            }

            Object keyLengthRandomNumber = options.get(KEY_LENGTH_RANDOM_NUMBER);

            if (keyLengthRandomNumber == null) {
                keyLengthRandomNumber = Integer.valueOf(0);
            }

            this.secureRandomProvider = new DefaultSecureRandomProvider(
                saltAlgorithm.toString(),
                Integer.valueOf(keyLengthRandomNumber.toString()));
        }
    }

    this.secureRandom = createSecureRandom();
}
```

The credential validation logic is defined by the **`validateCredential()`** method. This method (which the parent **`AbstractCredentialHandler`** class declares as an abstract method) checks the validity of the credential value passed in and either returns **`true`** if the credential is valid or **`false`** if it is not. The **`validateCredential()`** method is a convenience method which delegates much of the boilerplate code required for credential validation to **`AbstractCredentialHandler`**, allowing the subclass to simply define the bare minimum code required to validate the credential. If you were to implement a **`CredentialHandler`** without using **`AbstractCredentialHandler`** as a base class, you would instead need to implement the **`validate()`** method which in general requires a fair bit more code.

```
@Override
protected boolean validateCredential(final CredentialStorage storage,
    final V credentials) {
    EncodedPasswordStorage hash = (EncodedPasswordStorage) storage;

    if (hash != null) {
        String rawPassword = new String(credentials.getPassword().getValue());
        return this.passwordEncoder.verify(saltPassword(rawPassword,
            hash.getSalt()), hash.getEncodedHash());
    }

    return false;
```

```
}
```

The **`update()`** method (in contrast to **`validateCredential`**) *is* defined by the **`CredentialHandler`** interface itself, and is used to persist a credential value to the backend identity store. In **`PasswordCredentialHandler`** this method creates a new instance of **`EncodedPasswordStorage`**, a **`CredentialStorage`** implementation that represents a password's hash and salt values. The salt value in this implementation is randomly generated using the configured property values, and then used to encode the password hash. This value is then stored by calling the **`store.storeCredential()`** method.

```
@Override
public void update(IdentityContext context, Account account, U password, S store,
                   Date effectiveDate, Date expiryDate) {

    EncodedPasswordStorage hash = new EncodedPasswordStorage();

    if (password.getValue() == null || isNullOrEmpty(password.getValue().toString())) {
        throw MESSAGES.credentialInvalidPassword();
    }

    String rawPassword = new String(password.getValue());

    String passwordSalt = generateSalt();

    hash.setSalt(passwordSalt);
    hash.setEncodedHash(this.passwordEncoder.encode(saltPassword(rawPassword,
        passwordSalt)));

    if (effectiveDate != null) {
        hash.setEffectiveDate(effectiveDate);
    }

    hash.setExpiryDate(expiryDate);

    store.storeCredential(context, account, hash);
}
```

Report a bug[13]

4.6. Validating Credentials for Custom Account Types

The built-in credential types use the **`Account`** types provided by the Basic Model when validating or updating credentials. That said, only the following types can be used with the built-in credential types, by default:

- **`org.picketlink.idm.model.basic.Agent`**
- **`org.picketlink.idm.model.basic.User`**

These are the **`Account`** types provided by the Basic Model.

[13] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29696-675027+%5BLatest%5D&comment=Title%3A+Implementing+a+Custom+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3ECredentialHandler%3C%2Fcode%3E%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29696-675027+24+Jun+2014+00%3A20+en-US+%5BLatest%5D

As previously discussed, PicketLink provides a very flexible Identity Model, from which you can build your own model with your own types. You may decide to use none of these **`Account`** types and use your own to better represent your users.

Let's say you have a custom **`Account`** type called **`MyUser`**. Which may look like this:

```
@IdentityStereotype(USER)
public class MyUser extends AbstractIdentityType implements Account {

   @AttributeProperty
   @Unique
   @StereotypeProperty(IDENTITY_USER_NAME)
   private String loginName;

    // getters and setters
}
```

If you try to update or validate a password-based credential (which is one of the built-in types) using this type, PicketLink will not be able to perform these operations because this type is not known.

To let PicketLink aware about your custom **`Account`** types you must provide them during the configuration as follows:

```
IdentityConfigurationBuilder builder = event.getConfig();

        builder
            .named("default.config")
                .stores()
                    .jpa()
                        .supportType(MyUser.class)
```

You may notice that **`MyUser`** is annotated with **`@IdentityStereotype(USER)`** and also defines a **`loginName`** property annotated with **`@StereotypeProperty(IDENTITY_USER_NAME)`** to represent the user name. Those annotations are important to tell PicketLink that your type represents an user and the **`loginName`** property is used to store his name. The latter is going to be used to retrieve the account from the underlying stores when updating or validating credentials.

Report a bug[14]

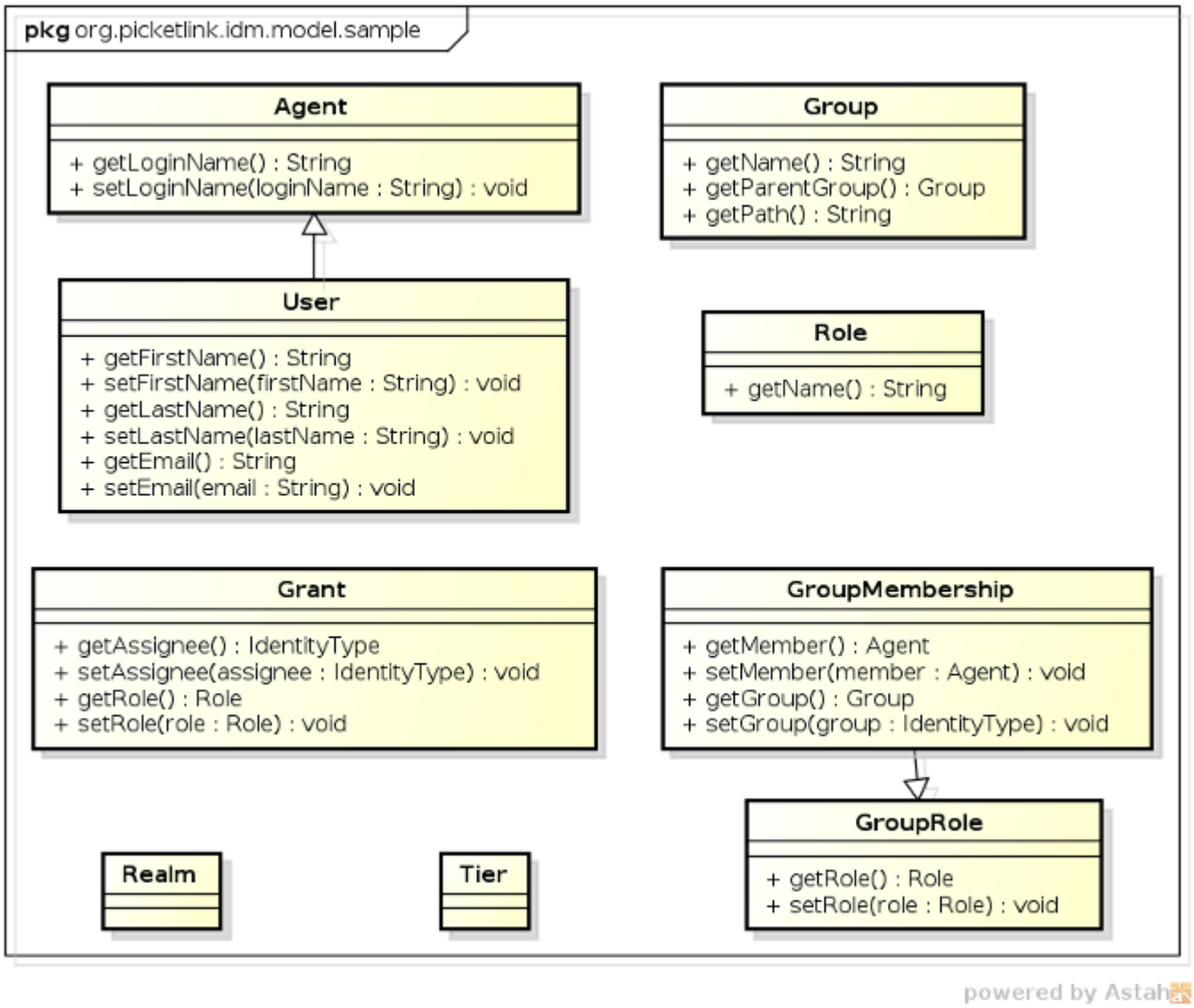
[14] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30406-666489+%5BLatest%5D&comment=Title%3A+Validating+Credentials+for+Custom+Account+Types%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30406-666489+11+Jun+2014+07%3A05+en-US+%5BLatest%5D

Identity Management - Basic Identity Model

5.1. Basic Identity Model

For the sake of convenience, PicketLink provides a basic identity model that consists of a number of core interfaces which define a set of fundamental identity types which might be found in a typical application. The usage of this identity model is entirely optional; for an application with basic security requirements the basic identity model might be more than sufficient, however for a more complex application or application with custom security requirements it may be necessary to create a custom identity model.

The following class diagram shows the classes and interfaces in the **`org.picketlink.idm.model.basic`** package:



- **`Agent`** represents a unique entity that may access the services secured by PicketLink. In contrast to a user which represents a human, **`Agent`** is intended to represent a third party non-human (i.e.

machine to machine) process that may authenticate and interact with your application or services. It declares methods for reading and setting the **`Agent`**'s login name.

- **`User`** represents a human user that accesses your application and services. In addition to the login name property defined by its parent interface **`Agent`**, the **`User`** interface declares a number of other methods for managing the user's first name, last name and e-mail address.
- **`Group`** is used to manage collections of identity types. Each **`Group`** has a name and an optional parent group.
- **`Role`** is used in various relationship types to designate authority to another identity type to perform various operations within an application. For example, a forum application may define a role called *moderator* which may be assigned to one or more **`User`**s or **`Group`**s to indicate that they are authorized to perform moderator functions.
- **`Grant`** relationship represents the assignment of a **`Role`** to an identity.
- **`GroupMembership`** relationship represents a **`User`** (or **`Agent`**) membership within a **`Group`**.
- **`GroupRole`** relationship represents the the assignment of a specific **`Role`** within a **`Group`** to a **`User`** or **`Agent`**. The reason this relationship extends the **`GroupMembership`** relationship is simply so it inherits the **`getMember()`** and **`getGroup()`** methods - being assigned to a **`GroupRole`** *does not* mean that the **`User`** (or **`Agent`**) that was assigned the group role also becomes a member of the group.
- **`Realm`** is a partition type, and may be used to store any **`IdentityType`** objects (including **`Agent`**s, **`User`**s, **`Group`**s or **`Role`**s.
- **`Tier`** is a specialized partition type, and may be only used to store **`Group`** or **`Role`** objects specific to an application.

Report a bug[1]

5.1.1. Utility Class for the Basic Identity Model

PicketLink also provides an utility class with some very useful and common methods to manipulate the basic identity model. Along the documentation you'll find a lot of examples using the the following class: **`org.picketlink.idm.model.basic.BasicModel`**. If you're using the basic identity model, this helper class can save you a lot of code and make your application even more simple.

The list below summarizes some of the functionalities provided by this class:

- Retrieve **`User`** and **`Agent`** instances by login name.
- Retrieve **`Role`** and **`Group`** instances by name.
- Add users as group members, grant roles to users. As well check if an user is member of a group or has a specific role.

One import thing to keep in mind is that the **`BasicModel`** helper class is only suitable if you're using the types provided by the basic identity model, only. If you are using custom types, even if those are

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29697-675028+%5BLatest%5D&comment=Title%3A+Basic+Identity+Model%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

sub-types of any of the types provided by the basic identity model, you should handle those custom types directly using the PicketLink IDM API.

As an example, let's suppose you have a custom type which extends the **`Agent`** type.

```
SalesAgent salesAgent = BasicModel.getUser(identityManager, "someSalesAgent");
```

The code above will never return a **`SalesAgent`** instance. The correct way of doing that is using the Query API directly as follows:

```
public SaleAgent findSalesAgent(String loginName) {
    List<SaleAgent> result = identityManager
                             .createIdentityQuery(SaleAgent.class)
                             .setParameter(SaleAgent.LOGIN_NAME, loginName)
                             .getResultList();
    return result.isEmpty() ? null : result.get(0);
}
```

Warning

Please note that the **`BasicModel`** helper class is only suitable for use cases where only the types provided by the basic identity model are used. If your application have also custom types, they need to be handled directly using the PicketLink IDM API.

Report a bug[2]

5.2. Managing Users, Groups and Roles

PicketLink IDM provides a number of basic implementations of the identity model interfaces for convenience, in the **`org.picketlink.idm.model.basic`** package. The following sections provide examples that show these implementations in action.

Report a bug[3]

5.2.1. Managing Users

The following code example demonstrates how to create a new user with the following properties:

- Login name - *jsmith*
- First name - *John*

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29698-620304+%5BLatest%5D&comment=Title%3A+Utility+Class+for+the+Basic+Identity+Model%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29698-620304+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29699-620305+%5BLatest%5D&comment=Title%3A+Managing+Users%2C+Groups+and+Roles%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

- Last name - *Smith*
- E-mail - *jsmith@acme.com*

```
User user = new User("jsmith");
user.setFirstName("John");
user.setLastName("Smith");
user.setEmail("jsmith@acme.com");
identityManager.add(user);
```

Once the **`User`** is created, it's possible to look it up using its login name:

```
User user = BasicModel.getUser(identityManager, "jsmith");
```

User properties can also be modified after the User has already been created. The following example demonstrates how to change the e-mail address of the user we created above:

```
User user = BasicModel.getUser(identityManager, "jsmith");
user.setEmail("john@smith.com");
identityManager.update(user);
```

Users may also be deleted. The following example demonstrates how to delete the user previously created:

```
User user = BasicModel.getUser(identityManager, "jsmith");
identityManager.remove("jsmith");
```

Report a bug[4]

5.2.2. Managing Groups

The following example demonstrates how to create a new group called *employees*:

```
Group employees = new Group("employees");
```

It is also possible to assign a parent group when creating a group. The following example demonstrates how to create a new group called *managers*, using the *employees* group created in the previous example as the parent group:

```
Group managers = new Group("managers", employees);
```

To lookup an existing **`Group`**, the **`getGroup()`** method may be used. If the group name is unique, it can be passed as a single parameter:

```
Group employees = BasicModel.getGroup(identityManager, "employees");
```

If the group name is not unique, the parent group must be passed as the second parameter (although it can still be provided if the group name is unique):

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29700-626775+%5BLatest%5D&comment=Title%3A+Managing+Users%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29700-626775+01+Apr+2014+20%3A48+en-US+%5BLatest%5D

```
Group managers = BasicModel.getGroup(identityManager, "managers", employees);
```

It is also possible to modify a **`Group`**'s name and other properties (besides its parent) after it has been created. The following example demonstrates how to disable the "employees" group we created above:

```
Group employees = BasicModel.getGroup(identityManager, "employees");
employees.setEnabled(false);
identityManager.update(employees);
```

To remove an existing group, we can use the **`remove()`** method:

```
Group employees = BasicModel.getGroup(identityManager, "employees");
identityManager.remove(employees);
```

Report a bug[5]

5.3. Managing Relationships

Relationships are used to model *typed associations* between two or more identities. All concrete relationship types must implement the marker interface **`org.picketlink.idm.model.Relationship`**:

The **`RelationshipManager`** interface provides three standard methods for managing relationships:

```
void add(Relationship relationship);
void update(Relationship relationship);
void remove(Relationship relationship);
```

- The **`add()`** method is used to create a new relationship.
- The **`update()`** method is used to update an existing relationship.

Note

Please note that the identities that participate in a relationship cannot be updated themselves, however the attribute values of the relationship can be updated. If you absolutely need to modify the identities of a relationship, then delete the relationship and create it again.

- The **`remove()`** method is used to remove an existing relationship.
- The **`createRelationshipQuery()`** method is used to perform relationship queries.
- The **`inheritsPrivileges()`** method is used to check whether an identity inherits privileges from another.

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29701-620307+%5BLatest%5D&comment=Title%3A+Managing+Groups%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29701-620307+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

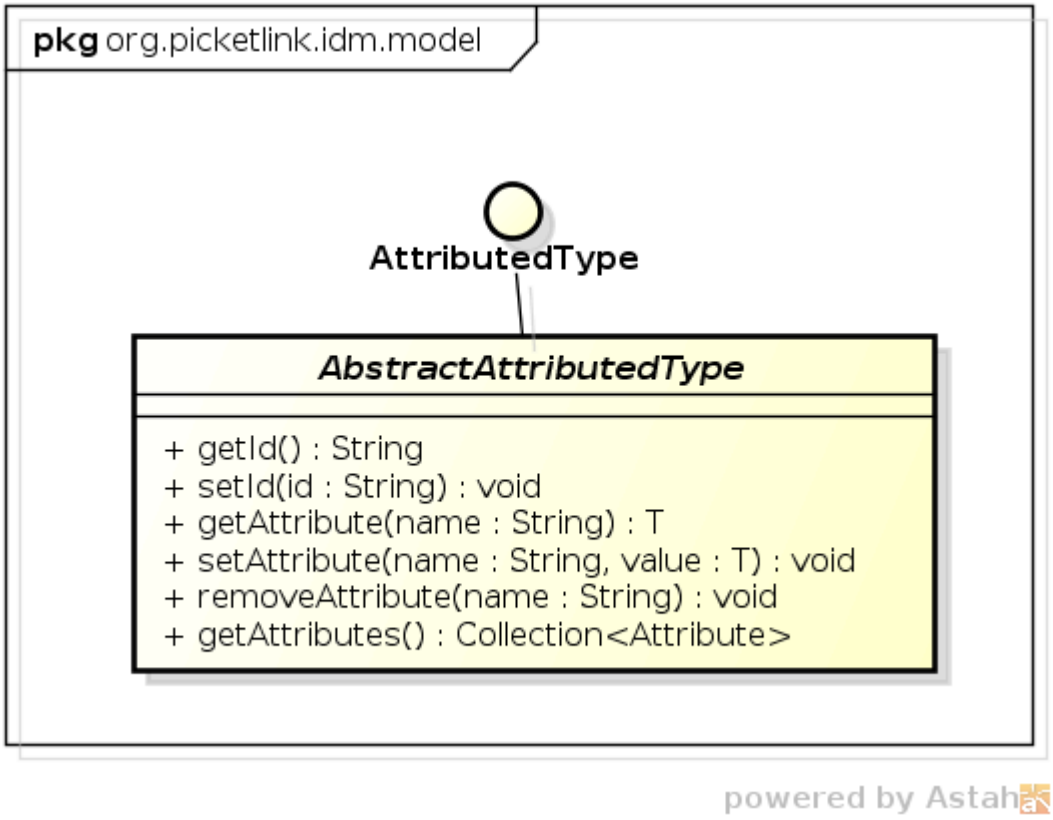
Note

To search for existing relationships between identity objects, use the Relationship Query API described later in this chapter.

Report a bug[6]

5.3.1. Built In Relationship Types

PicketLink provides a number of built-in relationship types, designed to address the most common requirements of a typical application. The following sections describe the built-in relationships and how they are intended to be used. Every built-in relationship type extends the **`AbstractAttributedType`** abstract class, which provides the basic methods for setting a unique identifier value and managing a set of attribute values:



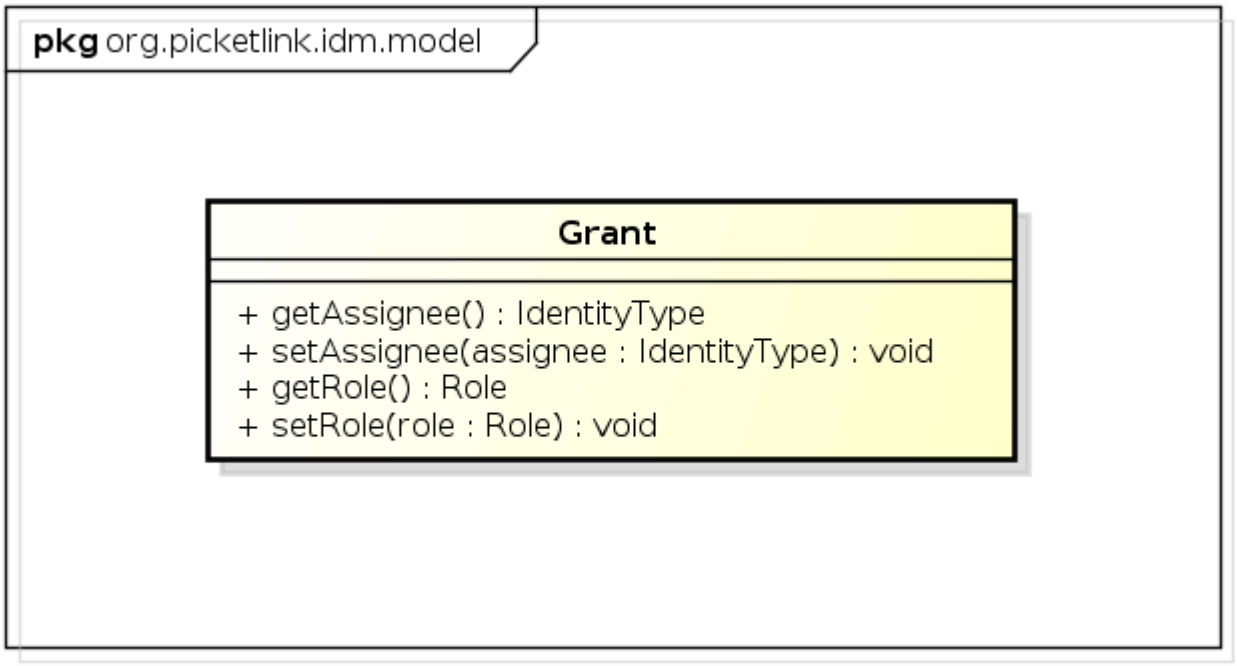
What this means in practical terms, is that every single relationship is assigned and can be identified by, a unique identifier value. Also, arbitrary attribute values may be set for all relationship types, which is useful if you require additional metadata or any other type of information to be stored with a relationship.

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29702-675029+%5BLatest%5D&comment=Title%3A+Managing+Relationships%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

Report a bug[7]

5.3.1.1. Application Roles

Application roles are represented by the **`Grant`** relationship, which is used to assign application-wide privileges to a **`User`** or **`Agent`**.



The **`RelationshipManager`** interface provides methods for directly granting a role. Here's a simple example:

```
User bob = BasicModel.getUser(identityManager, "bob");
Role superuser = BasicModel.getRole(identityManager, "superuser");
BasicModel.grantRole(relationshipManager, bob, superuser);
```

The above code is equivalent to the following:

```
User bob = BasicModel.getUser(identityManager, "bob");
Role superuser = BasicModel.getRole(identityManager, "superuser");
Grant grant = new Grant(bob, superuser);
identityManager.add(grant);
```

A granted role can also be revoked using the **`revokeRole()`** method:

```
User bob = BasicModel.getUser(identityManager, "bob");
Role superuser = BasicModel.getRole(identityManager, "superuser");
BasicModel.revokeRole(relationshipManager, bob, superuser);
```

To check whether an identity has a specific role granted to them, we can use the **`hasRole()`** method:

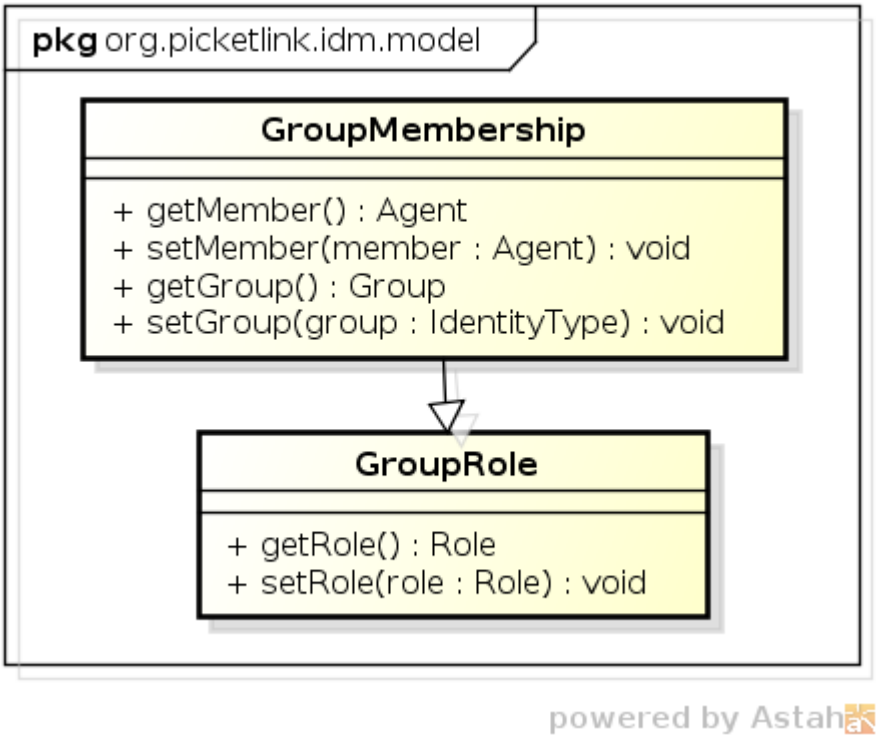
[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29703-675030+%5BLatest%5D&comment=Title%3A+Built+In+Relationship+Types%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
User bob = BasicModel.getUser(identityManager, "bob");
Role superuser = BasicModel.getRole(identityManager, "superuser");
boolean isBobASuperUser = BasicModel.hasRole(relationshipManager, bob, superuser);
```

Report a bug[8]

5.3.1.2. Groups and Group Roles

The **`GroupMembership`** and **`GroupRole`** relationships are used to represent a user's membership within a **`Group`**, and a user's role for a group, respectively.



Note

While the **`GroupRole`** relationship type extends **`GroupMembership`**, it does *not* mean that a member of a **`GroupRole`** automatically receives **`GroupMembership`** membership also - these are two distinct relationship types with different semantics.

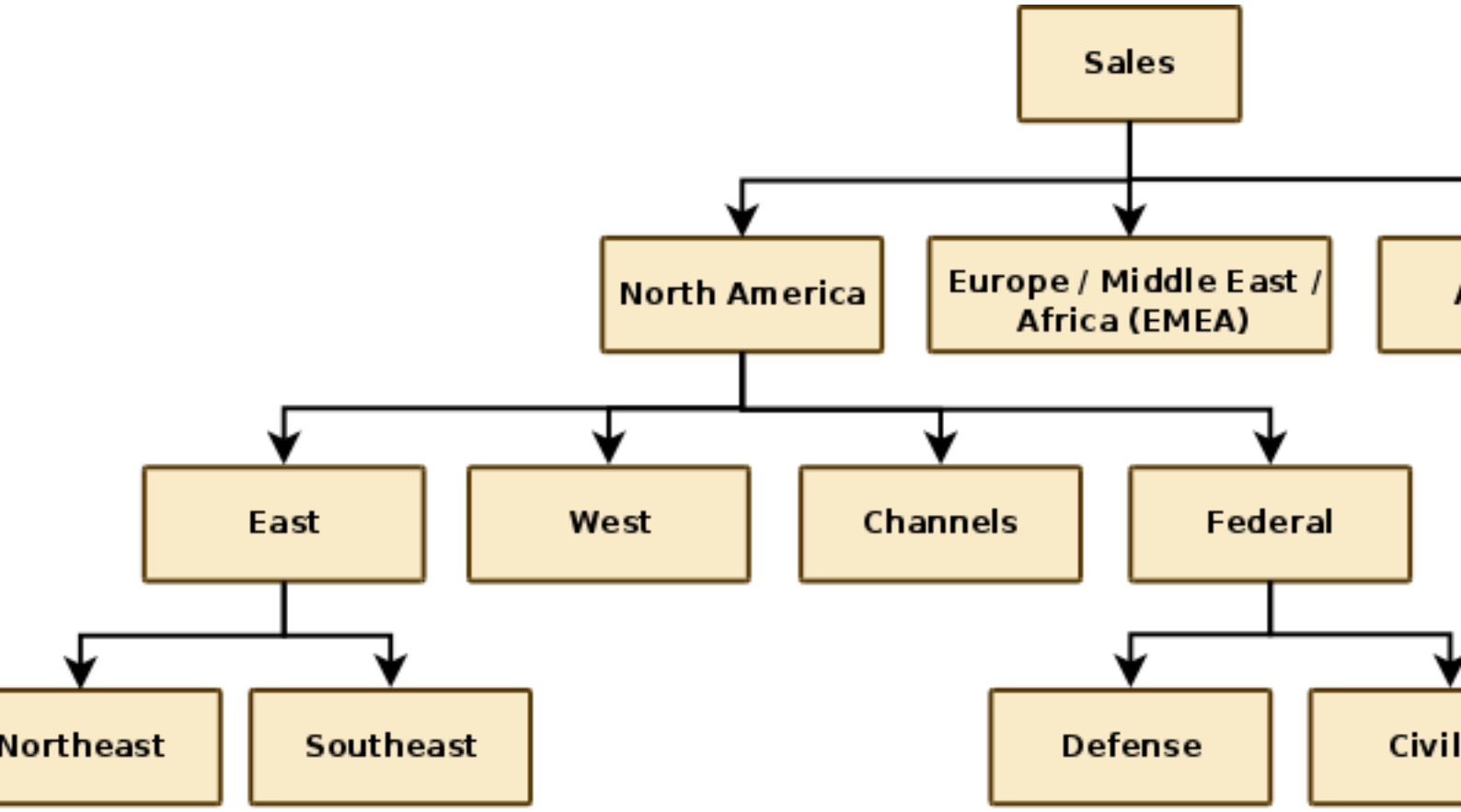
A **`Group`** is typically used to form logical collections of users. Within an organisation, groups are often used to mirror the organisation's structure. For example, a corporate structure might consist of a sales department, administration, management, etc. This structure can be modelled in PicketLink by creating corresponding groups such as *sales*, *administration*, and so forth. Users (who would represent the employees in a corporate structure) may then be assigned group memberships

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29704-675032+%5BLatest%5D&comment=Title%3A+Application+Roles%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29704-675032+24+Jun+2014+00%3A28+en-US+%5BLatest%5D

corresponding to their place within the company's organisational structure. For example, an employee who works in the sales department may be assigned to the *sales* group. Specific application privileges can then be blanket assigned to the *sales* group, and anyone who is a member of the group is free to access the application's features that require those privileges.

The **`GroupRole`** relationship type should be used when it is intended for an identity to perform a specific role for a group, but not be an actual member of the group itself. For example, an administrator of a group of doctors may not be a doctor themselves, but have an administrative role to perform for that group. If the intent is for an individual identity to both be a member of a group *and* have an assigned role in that group also, then the identity should have both **`GroupRole`** and **`GroupMembership`** relationships for that group.

Let's start by looking at a simple example - we'll begin by making the assumption that our organization is structured in the following way:



The following code demonstrates how we would create the hypothetical *Sales* group which is displayed at the head of the above organisational chart:

```
Group sales = new Group("Sales");
identityManager.add(sales);
```

We can then proceed to create its subgroups:

```
identityManager.add(new Group("North America", sales);
identityManager.add(new Group("EMEA", sales);
identityManager.add(new Group("Asia", sales);
// and so forth
```

The second parameter of the **`Group()`** constructor is used to specify the group's parent group. This allows us to create a hierarchical group structure, which can be used to mirror either a simple or

complex personnel structure of an organisation. Let's now take a look at how we assign users to these groups.

The following code demonstrates how to assign an *administrator* group role for the *Northeast* sales group to user *jsmith*. The *administrator* group role may be used to grant certain users the privilege to modify permissions and roles for that group:

```
Role admin = BasicModel.getRole(identityManager, "administrator");
User user = BasicModel.getUser(identityManager, "jsmith");
Group group = BasicModel.getGroup(identityManager, "Northeast");
BasicModel.grantGroupRole(relationshipManager, user, admin, group);
```

A group role can be revoked using the **`revokeGroupRole()`** method:

```
BasicModel.revokeGroupRole(relationshipManager, user, admin, group);
```

To test whether a user has a particular group role, you can use the **`hasGroupRole()`** method:

```
boolean isUserAGroupAdmin = BasicModel.hasGroupRole(relationshipManager, user, admin, group);
```

Next, let's look at some examples of how to work with simple group memberships. The following code demonstrates how we assign sales staff *rbrown* to the *Northeast* sales group:

```
User user = BasicModel.getUser(identityManager, "rbrown");
Group group = BasicModel.getGroup(identityManager, "Northeast");
BasicModel.addToGroup(relationshipManager, user, group);
```

A **`User`** may also be a member of more than one **`Group`**; there are no built-in limitations on the number of groups that a **`User`** may be a member of.

We can use the **`removeFromGroup()`** method to remove the same user from the group:

```
BasicModel.removeFromGroup(relationshipManager, user, group);
```

To check whether a user is the member of a group we can use the **`isMember()`** method:

```
boolean isUserAMember = BasicModel.isMember(relationshipManager, user, group);
```

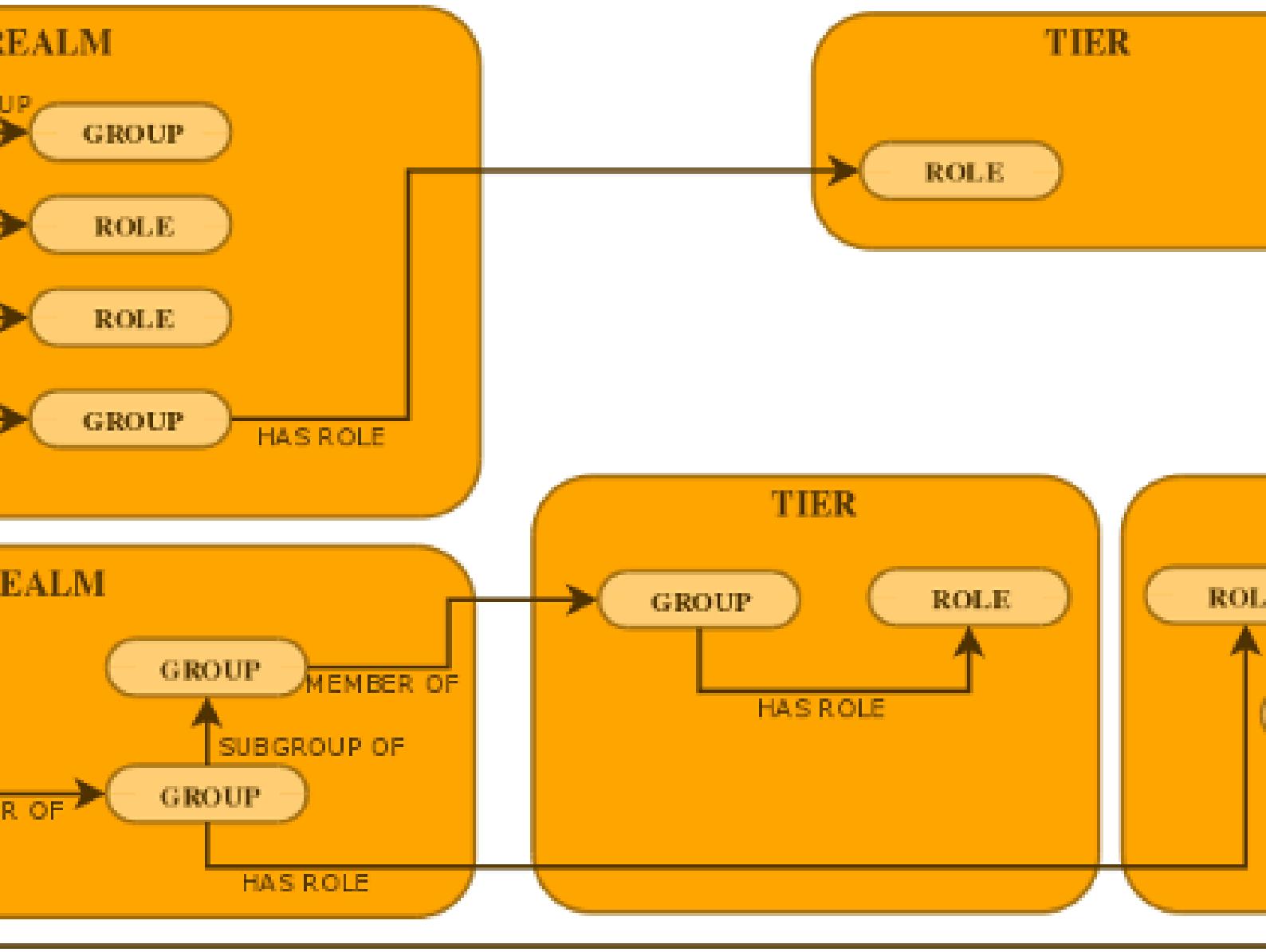
Relationships can also be created via the **`add()`** method. The following code is equivalent to assigning a group role via the **`grantGroupRole()`** method shown above:

```
Role admin = BasicModel.getRole(identityManager, "administrator");
User user = BasicModel.getUser(identityManager, "jsmith");
Group group = BasicModel.getGroup(identityManager, "Northeast");
GroupRole groupRole = new GroupRole(user, group, admin);
identityManager.add(groupRole);
```

Report a bug[9]

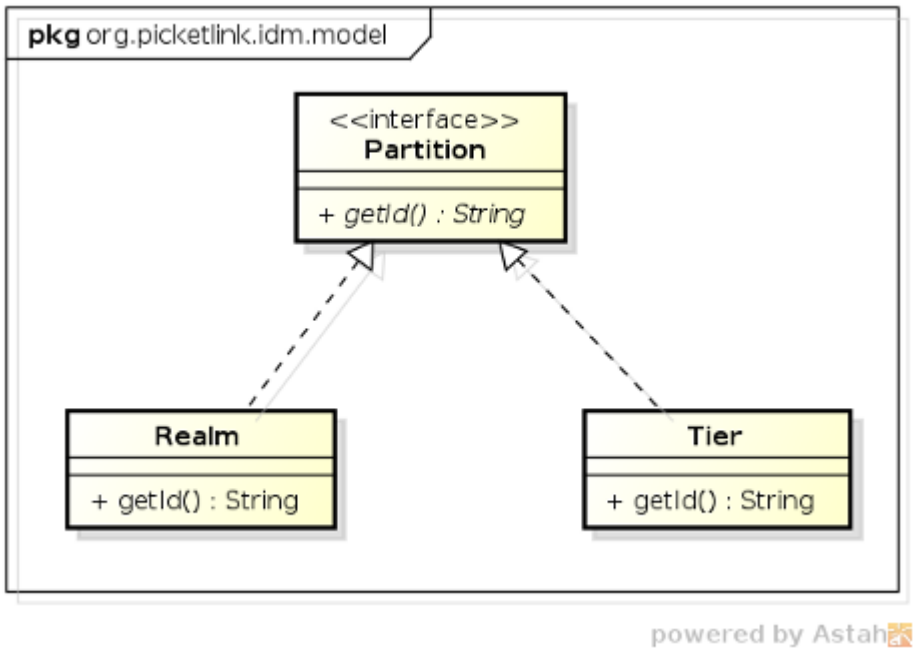
[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29705-675033+%5BLatest%5D&comment=Title%3A+Groups+and+Group+Roles%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for

5.4. Realms and Tiers



In terms of API, both the **`Realm`** and **`Tier`** classes implement the **`Partition`** interface, as shown in the following class diagram:

+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29705-675033+24+Jun+2014+00%3A30+en-US+%5BLatest%5D



A *Realm* is used to define a discrete set of users, groups and roles. A typical use case for realms is the segregation of corporate user accounts within a multi-tenant application, although it is not limited this use case only. As all identity management operations must be performed within the context of an *active partition*, PicketLink defines the concept of a *default realm* which becomes the active partition if no other partition has been specified.

A *Tier* is a more restrictive type of partition than a realm, as it only allows groups and roles to be defined (but not users). A Tier may be used to define a set of application-specific groups and roles, which may then be assigned to groups within the same Tier, or to users and groups within a separate Realm.

Report a bug[10]

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29706-675034+%5BLatest%5D&comment=Title%3A+Realms+and+Tiers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29706-675034+24+Jun+2014+00%3A32+en-US+%5BLatest%5D

Identity Management - Attribute Management

6.1. Overview

PicketLink IDM classifies attributes in two main categories: *formal* and *ad-hoc* attributes.

Usually all attributes you need for a specific type is mapped to its properties. We call those attributes *formal attributes.*. Formal attributes are properties managed by PicketLink and defined directly in your types. By managed, we mean that PicketLink knows how to get the value to store from a specific property and also retrieve its value from the store and populate back to an instance of your type.

But sometimes you may need to define attributes that are not mapped to a specific property of your type. Specially if those are dynamic attributes, whose existence depends on a specific context or rule. We call those attributes *ad-hoc attributes*. Ad-hoc attributes are not strong-typed as formal attributes, they are just a key/value pair. Where the key is the attribute's name and the latter its value.

Report a bug[1]

6.2. Formal attributes

Usually all attributes are formal attributes, in the sense that properties of a type are managed by PicketLink. In order to get them managed by PicketLink you need to annotate each property of your type with *Section 3.5.1, "The `@AttributeProperty` Annotation"*

If you take the **`User`** type as an example, you'll see that all its properties are defined as follows:

```
public class User extends Agent {

  @AttributeProperty
  private String firstName;

  @AttributeProperty
  private String lastName;

  @AttributeProperty
  private String email;

}
```

Formal attributes are strongly-typed as they are directly defined as properties in your type.

Those attributes are also queriable. Which means you can use the Query API to search for types with a specific property and value. If a property is queriable, we recommend to always create a **`QueryParameter`** constant as follows:

```
public class User extends Agent {
```

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29707-620313+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29707-620313+12+Mar+2014+12%3A30+en-US+%5BLatest%5D

```
  /**
   * A query parameter used to set the firstName value.
   */
  public static final QueryParameter FIRST_NAME = QUERY_ATTRIBUTE.byName("firstName");

  /**
   * A query parameter used to set the lastName value.
   */
  public static final QueryParameter LAST_NAME = QUERY_ATTRIBUTE.byName("lastName");

  /**
   * A query parameter used to set the email value.
   */
  public static final QueryParameter EMAIL = QUERY_ATTRIBUTE.byName("email");

  @AttributeProperty
  private String firstName;

  @AttributeProperty
  private String lastName;

  @AttributeProperty
  private String email;

}
```

Once the query parameters are defined and mapped to your properties, you can search for types based on its properties as follows:

```
IdentityQuery<User> query = identityManager.<User> createIdentityQuery(User.class);

  query.setParameter(User.FIRST_NAME, "John");

  // find only by the first name
  List<User> result = query.getResultList();

  for (User user : result) {
    // do something
  }
```

Report a bug[2]

6.3. Ad-hoc attributes

The best way to understand ad-hoc attributes is look how they're defined. The example below uses three attributes to define some security questions for an user.

```
User user = new User("john");

  user.setAttribute(new Attribute<Integer>("QuestionTotal", 2);

  // attribute for question #1
  user.setAttribute(new Attribute<String>("Question1", "What is favorite toy?"));
  user.setAttribute(new Attribute<String>("Question1Answer", "Gum"));

  // attribute for question #2
```

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29708-620662+%5BLatest%5D&comment=Title%3A+Formal+attributes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29708-620662+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

```
  user.setAttribute(new Attribute<String>("Question2", "What is favorite word?"));
  user.setAttribute(new Attribute<String>("Question2Answer", "Hi"));

identityManager.add(user);
```

Ad-hoc attributes are not strong-typed as formal attributes, they are just a key/value pair. Where the key is the attribute's name and the latter its value. The key point here is that they offer great flexibility to your identity model as you can define any attribute you want without changing any type.

Note

Note that ad-hoc attributes are not typed. You should use them wisely, otherwise they may become unmanageable.

In theory, all PicketLink types such as **`IdentityType`**, **`Relationship`** and **`Partition`** support ad-hoc attributes. The reason is that all those types are a subtypes of **`AttributedType`**. But in practice, even if a type support ad-hoc attributes the underlying store may not. Which means you can not use those attributes. The LDAP Identity Store, for example, does not support ad-hoc attributes at all if used *alone*. Take a look at *Chapter 9, Identity Management - Working with LDAP* for more details about how to support ad-hoc attributes when using the LDAP store.

Those attributes are also queriable. The example below shows how you can use the Query API to search for types using ad-hoc attributes:

```
IdentityQuery<User> query = identityManager.<User> createIdentityQuery(User.class);


 query.setParameter(IdentityType.QUERY_ATTRIBUTE.byName("SomeAttribute"), "SomeAttributeValue");

  List<User> result = query.getResultList();

  for (User user : result) {
    // do something
  }
```

Report a bug[3]

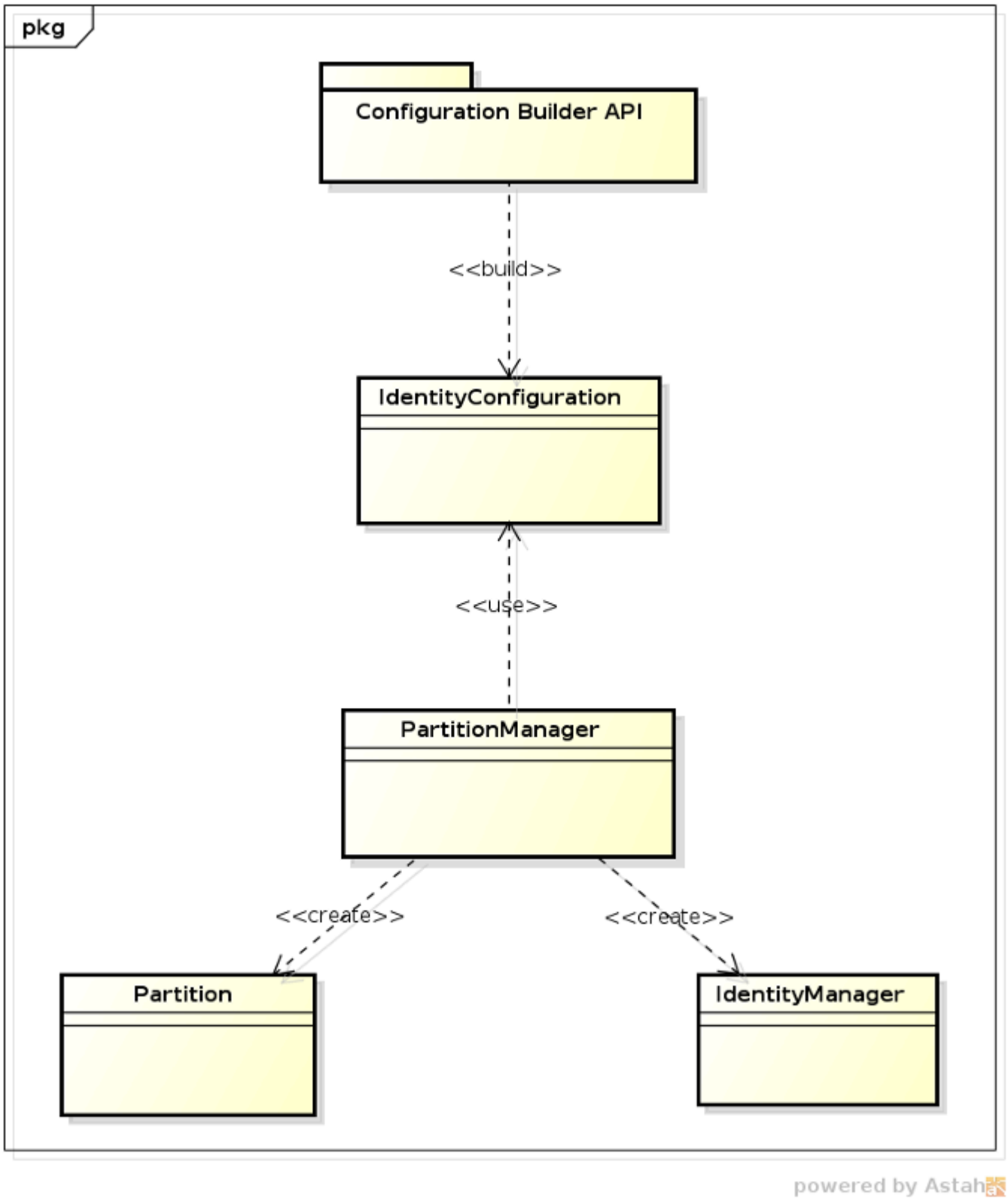
[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29709-620663+%5BLatest%5D&comment=Title%3A+Ad-hoc+attributes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29709-620663+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

Identity Management - Configuration

7.1. Configuration

7.1.1. Architectural Overview

Configuration in PicketLink is in essence quite simple; an **`IdentityConfiguration`** object must first be created to hold the PicketLink configuration options. Once all configuration options have been set, you just create a **`PartitionManager`** instance passing the previously created configuration. The **`PartitionManager`** can then be used to create **`Partition`** and **`IdentityManager`** instances.



The **`IdentityConfiguration`** is usually created using a Configuration Builder API, which provides a rich and fluent API for every single aspect of PicketLink configuration.

Report a bug[1]

7.1.2. Default Configuration

If you'd like to get up and running with IDM quickly, the good news is that PicketLink will provide a default configuration that stores your identity data on the file system if no other configuration is available. This means that if you have the PicketLink libraries in your project, you can simply inject the **`PartitionManager`**, **`IdentityManager`** or **`RelationshipManager`** beans into your own application and start using them immediately:

```
@Inject PartitionManager partitionManager;
@Inject IdentityManager identityManager;
@Inject RelationshipManager relationshipManager;
```

Note

The default configuration is very useful for developing and testing purposes, as you don't need a database or a LDAP server to start managing your identity data.

Report a bug[2]

7.1.3. Providing a Custom Configuration

In certain cases the default configuration may not be enough to your application. You can easily provide your own configuration by observing a specific **`IdentityConfigurationEvent`**:

```
public class IdentityManagementConfiguration {

  public void observeIdentityConfigurationEvent(@Observes IdentityConfigurationEvent event) {
    IdentityConfigurationBuilder builder = event.getConfig();

    // use the builder to provide your own configuration
  }
}
```

You can also provide your own configuration by producing one or more **`IdentityConfiguration`** instances using a **`@Producer`** annotated method:

```
public class IdentityManagementConfiguration {

  @Produces
```

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29710-675036+%5BLatest%5D&comment=Title%3A+Architectural+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29710-675036+24+Jun+2014+00%3A34+en-US+%5BLatest%5D

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29711-620317+%5BLatest%5D&comment=Title%3A+Default+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29711-620317+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

```
  public IdentityConfiguration produceJPAConfiguration() {
      IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

      builder
          .named("jpa.config")
              .stores()
                  .jpa()
                    .supportAllFeatures()

      return builder.build();
  }

  @Produces
  public IdentityConfiguration produceLDAPConfiguration() {
      IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

      builder
          .named("ldap.config")
              .stores()
                  .ldap()
                      // configure the LDAP store

      return builder.build();
  }
}
```

The example above produces two distinct configurations: one using a JPA store and another using the LDAP store. During the startup PicketLink will resolve both configurations and initialize the IDM subsystem with them. You can also provide a single configuration.

Warning

When producing **`IdentityConfiguration`** instances the produced bean must be dependent and not normal-scoped.

For last, you can also build your own **`PartitionManager`** instance if you want more control.

```
@ApplicationScoped
public static class PicketLinkConfiguration {

  @Inject
  private EntityManagerContextInitializer contextInitializer;

  @PicketLink
  @Produces
  public PartitionManager producePartitionManager() {
      IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

      builder
          .named("produced.partition.manager.config")
              .stores()
                  .jpa()
                      .mappedEntity(
                              AccountTypeEntity.class,
                              RoleTypeEntity.class,
                              GroupTypeEntity.class,
                              IdentityTypeEntity.class,
                              RelationshipTypeEntity.class,
                              RelationshipIdentityTypeEntity.class,
                              PartitionTypeEntity.class,
```

```
                                PasswordCredentialTypeEntity.class,
                                DigestCredentialTypeEntity.class,
                                X509CredentialTypeEntity.class,
                                OTPCredentialTypeEntity.class,
                                AttributeTypeEntity.class,
                                TokenCredentialTypeEntity.class
                        )
                        .addContextInitializer(this.contextInitializer)
                        .supportAllFeatures();

        PartitionManager partitionManager = new DefaultPartitionManager(builder.build());

        Partition defaultPartition = new Realm(Realm.DEFAULT_REALM);

        partitionManager.add(defaultPartition); // creates the default partition

        return partitionManager;
    }

}
```

The example above allows you to produce your own **`PartitionManager`** instance. Note that the producer method is annotated with the **`PicketLink`** annotation.

Important

When producing your own **`PartitionManager`** is that you must manually create the partitions before start producing **`IdentityManager`** instances (eg.: the default partition)

For last, you can also provide the configuration by observing the **`SecurityConfigurationEvent`** as follows:

```
public class IdentityManagementConfiguration {

    public void configureIdentityManagement(@Observes SecurityConfigurationEvent event) {
        SecurityConfigurationBuilder builder = event.getBuilder();

        builder
            .idmConfig()
                .named("default.config")
                    .stores()
                        .jpa()
                            .supportAllFeatures();
    }
}
```

You may be asking which approach is recommended, right ? We always recommend to use the last approach, observe the **`SecurityConfigurationEvent`**. The reason is because from this event you have access to not only the IDM config but for others features provided by PicketLink.

However you may want to provide your own **`PartitionManager`** instance, instead of relying on PicketLink to create one for you. In this case you can produce a **`PartitionManager`** by your own. Or if you want different methods for each IDM config you may want to produce **`IdentityConfiguration`** instances instead.

Report a bug[3]

7.1.4. Initializing the `PartitionManager`

You may need to initialize the **`PartitionManager`** with some data before your application starts to produce partition manager instances. PicketLink provides a specific event called **`PartitionManagerCreateEvent`**, which can be used to provide any initialization logic right after a **`PartitionManager`** instance is created and before it is consumed by any injection point in your application.

```
public class MyPartitionManagerInitializer {

    public void init(@Observes PartitionManagerCreateEvent event) {
        // retrieve the recently created partition manager instance
        PartitionManager partitionManager = event.getPartitionManager();

        // retrieve all the configuration used to build the instance
        Collection configurations = partitionManager.getConfigurations();
    }
}
```

One important thing to keep in mind when providing a observer for **`PartitionManagerCreateEvent`** is that if any partition is created during the initialization, PicketLink won't try to create the default partition.

Note

Apache TomEE users should always provide an observer for **`PartitionManagerCreateEvent`** in order to initialize the partition manager properly. Specially if using the JPA store and if no active transaction exists when the **`PartitionManager`** is being created.

Report a bug[4]

7.1.5. Programmatic Configuration Overview

The Identity Management configuration can be defined programmatically using the Configuration Builder API. The aim of this API is to make it easier to chain coding of configuration options in order to speed up the coding itself and make the configuration more *readable*.

Let's assume that you want to quick start with PicketLink Identity Management features using a File-based Identity Store. First, a fresh instance of **`IdentityConfiguration`** is created using the

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29712-714348+%5BLatest%5D&comment=Title%3A+Providing+a+Custom+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29712-714348+01+Oct+2014+06%3A30+en-US+%5BLatest%5D

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29713-680717+%5BLatest%5D&comment=Title%3A+Initializing+the+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EPartitionManager%3C%2Fcode%3E%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29713-680717+02+Jul+2014+06%3A48+en-US+%5BLatest%5D

IdentityConfigurationBuilder helper object, from where we choose which identity store we want to use (in this case a file-based store) and any other configuration option, if necessary. Finally, we use the configuration to create a **PartitionManager**, from where we can create **Partition** and **IdentityManager** instances:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
  .named("default")
    .stores()
      .file()
        .supportAllFeatures();

DefaultPartitionManager partitionManager = new DefaultPartitionManager(builder.buildAll());

Realm defaultRealm = new Realm(Realm.DEFAULT_REALM);

// let's add the partition using the default configuration.
partitionManager.add(defaultRealm);

// if no partition is specified to createIdentityManager, defaults to the default Realm.
IdentityManager identityManager = partitionManager.createIdentityManager();

User john = new User("john");

// let's add john to the default partition
identityManager.add(user);
```

Report a bug[5]

7.1.6. Providing Multiple Configurations

A **PartitionManager** can be built considering multiple configurations. This is a very powerful feature given that you can manage your identity data between different partitions each one using a specific configuration.

As discussed before, each configuration has a name. The name can be used to identify a configuration set as well to tell PicketLink the configuration that should be used to manage a specific **Partition**.

Let's take a more close look how you can use multiple configurations:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
    .named("ldap.config")
        .stores()
            .ldap()
              // specific configuration options for the LDAP store
              .supportAllFeatures();
    .named("jpa.config")
        .stores()
            .jpa()
              // specific configuration options for the JPA store
              .supportAllFeatures();
```

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29714-620320+%5BLatest%5D&comment=Title%3A+Programmatic+Configuration+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29714-620320+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

```
DefaultPartitionManager partitionManager = new DefaultPartitionManager(builder.buildAll());

Realm internalPartition = new Realm("internal");

// the 'internal' partition will use the 'ldap.config' configuration
partitionManager.add(internalPartition, "ldap.config");

// we create an IdentityManager for the LDAP managed partition
IdentityManager internalIdentityManager =
 partitionManager.createIdentityManager(internalPartition);

User john = new User("john");

// john will be added to the LDAP
internalIdentityManager.add(john);

Realm externalPartition = new Realm("external");

// the 'external' partition will use the 'jpa.config' configuration
partitionManager.add(externalPartition, "jpa.config");

User mary = new User("mary");

// we create an IdentityManager for the JPA managed partition
IdentityManager externalIdentityManager =
 partitionManager.createIdentityManager(externalPartition);

// mary will be added to the database
externalIdentityManager.add(mary);
```

The example above is just one of the different things you can do with PicketLink. The code above defines two partitions: one for internal users and another one for external users. Each partition is associated with one of the provided configurations where the internal partition will use LDAP to store users whether the external partition will use JPA.

When you create a **`IdentityManager`** for one of those partitions, all identity management operations will be done considering the configuration associated with the current partition. In other words, considering the example above, the user 'john' will be stored in the LDAP and 'mary' in a Database.

Report a bug[6]

7.1.7. Providing Multiple Stores for a Configuration

It is also possible to use multiple **`IdentityStore`** configurations in a single named configuration. This can be very useful when your identity data is distributed in different stores or even if a specific store have any kind of limitation that can be provided by another one.

For instance, the LDAP store have some limitations and does not support all features provided by PicketLink. One of those unsupported features is the ability to handle ad-hoc attributes. When using LDAP you're tied with a schema that usually is very hard to change in order to support all your needs.

In this cases, PicketLink allows you to combine in a single configuration the LDAP and the JPA store, for example. Where you can use LDAP for users, roles and groups and use the JPA store for relationships.

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29715-620321+%5BLatest%5D&comment=Title%3A+Providing+Multiple+Configurations%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29715-620321+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
  .named("default")
    .stores()
      .jpa()
        // configuration options for the jpa store
        .supportGlobalRelationship(Relationship.class)
        .supportAttributes(true)
      .ldap()
        // configuration options for the ldap store
        .supportType(IdentityType.class)
```

The example above defines a single configuration with two stores: LDAP and JPA. For the LDAP store configuration we define that only **`IdentityType`** types should be supported. In other words, we're only storing users, roles and groups. For the JPA store configuration we define that only **`Relationship`** types should be supported. In other words, we're only storing relationships such as **`Grant`**, **`GroupMembership`**, etc.

You may also notice that the JPA store is configured to support attributes too. What means that we can now use ad-hoc attributes for all the supported types.

Report a bug[7]

7.1.8. Configuring Credential Handlers

Each **`IdentityStore`** may support a different set of credential handlers. This documentations describes the built-in credential handlers provided by PicketLink, but sometimes you may want to provide your own implementations.

When you write your custom credential handler you need to tell PicketLink the identity store that will support it. This is done by the following code:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
  .named("default")
    .stores()
      .jpa()
        // other JPA configuration
        .addCredentialHandler(UserPasswordCredentialHandler.class)
        .supportAllFeatures();
```

The example above shows how to configure a credential handler for a JPA-based store using the **`addCredentialHandler`** method.

Report a bug[8]

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29716-620322+%5BLatest%5D&comment=Title%3A+Providing+Multiple+Stores+for+a+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29716-620322+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29717-620323+%5BLatest%5D&comment=Title%3A+Configuring+Credential+Handlers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

7.1.8.1. Passing parameters to Credential Handlers

Some credential handlers support a set of configuration options to configure their behavior. These options can be specified as follows:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
  .named("default")
    .stores()
      .jpa()
        // other JPA configuration

.setCredentialHandlerProperty(PasswordCredentialHandler.PASSWORD_ENCODER, new BCryptPasswordEncoder(4))
        .supportAllFeatures();
```

The example above shows how to set a property for the **`PasswordCredentialHandler`** using the **`setCredentialHandlerProperty`** method.

Report a bug[9]

7.1.9. Identity Context Configuration

The **`IdentityContext`** plays an important role in the PicketLink IDM achitecture. It is strongly used during the execution of operations. It carries very sensitive and contextual information for a specific operation and provides access for some of the IDM underlying services such as caching, event handling, id generator for **`AttributedType`** instances, among others.

Operations are always executed by a specific **`IdentityStore`** in order to persist or store identity data using a specific repository (eg.: LDAP, databases, filesystem, etc). When executing a operation the identity store must be able to:

- Access the current **`Partition`**. Eg.: **`Realm`** or **`Tier`**.
- Access the *Event Handling API* in order to fire events such as when an user is created, updated, etc.
- Access the *Caching API* in order to cache identity data and increase performance.
- Access to external resources, provided before the operation is executed and initialized by a **`ContextInitializer`**.

Report a bug[10]

7.1.9.1. Initializing the IdentityContext

Sometimes you may need to provide additional configuration or even references for external resources before the operation is executed by an identity store. An example is how you tell to

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29718-620324+%5BLatest%5D&comment=Title%3A+Passing+parameters+to+Credential+Handlers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29718-620324+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29719-620325+%5BLatest%5D&comment=Title%3A+Identity+Context+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

the **JPAIdentityStore** which **EntityManager** instance should be used. When executing an operation, the **JPAIdentityStore** must be able to access the current **EntityManager** to persist or retrieve data from the database. You need someway to populate the **IdentityContext** with such information. When you're configuring an identity store, there is a configuration option that allows you to provide a **ContextInitializer** implementation.

```
public interface ContextInitializer {
    void initContextForStore(IdentityContext context, IdentityStore<?> store);
}
```

The method **initContextForStore** will be invoked for every single operation and before its execution by the identity store. It can be implemented to provide all the necessary logic to initialize and populate the **IdentityContext** for a specific **IdentityStore**.

The configuration is also very simple:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
  .named("default")
    .stores()
      .file()
        .supportAllFeatures();
        .addContextInitializer(new MyContextInitializer());
```

You can provide multiple initializers.

Note

Remember that initializers are executed for every single operation. Also, the same instance is used between operations which means your implementation should be “stateless”. You should be careful about the implementation in order to not impact performance, concurrency or introduce unexpected behaviors.

Report a bug[11]

7.1.10. IDM configuration from XML file

Actually it's possible to configure IDM with XML configuration. This possibility is good especially in case when you want Picketlink IDM to be part of bigger system and your users won't have a possibility to change source code and so they can't configure it programmatically with the Builder API. So they will just need to change the configuration in XML file instead of doing some changes directly in source code.

[11] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29720-620326+%5BLatest%5D&comment=Title%3A+Initializing+the+IdentityContext%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29720-620326+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

Report a bug[12]

7.1.10.1. Unified XML configuration

Whole Picketlink project provides unified format of configuration file, so that you can configure *Section 14.1, "Overview"* and IDM in same file.

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">

  <PicketLinkIDP xmlns="urn:picketlink:identity-federation:config:1.0"
                 ServerEnvironment="tomcat" BindingType="POST" SupportsSignatures="true">
    <!-- SAML2 IDP configuration is here -->
  </PicketLinkIDP>

  <Handlers xmlns="urn:picketlink:identity-federation:handler:config:2.1">
     <!-- Configuration of SAML2 handlers is here -->
  </Handlers>

  <PicketLinkSTS xmlns="urn:picketlink:identity-federation:config:1.0"
                 STSName="Test STS" TokenTimeout="7200" EncryptToken="true">
    <!-- Configuration of Picketlink STS is here -->
  </PicketlinkSTS>

  <PicketLinkIDM>
    <!-- IDM configuration is here -->
  </PicketLinkIDM>

</PicketLink>
```

Note that if you don't want to use Picketlink Federation, you can omit it's configuration and use just IDM.

Report a bug[13]

7.1.10.2. XML configuration format

XML configuration is leveraging *Section 7.1.5, "Programmatic Configuration Overview"* and Java reflection during it's parsing, so names of XML elements are actually same like names of particular Builder methods.

For example, let's assume that you want to use **`FileIdentityStore`** and your programmatic configuration looks like this:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
    .named(SIMPLE_FILE_STORE_CONFIG)
        .stores()
            .file()
            .preserveState(false)
```

[12] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29721-620327+%5BLatest%5D&comment=Title%3A+IDM+configuration+from+XML+file%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[13] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29722-620664+%5BLatest%5D&comment=Title%3A+Unified+XML+configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29722-620664+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

```
            .supportGlobalRelationship(Relationship.class)
            .supportAllFeatures();
```

Same XML configuration to configure IDM with **`FileIdentityStore`** will look like this:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">

  <PicketLinkIDM>

    <named value="SIMPLE_FILE_STORE_CONFIG">
      <stores>
        <file>
          <preserveState value="false" />
          <supportGlobalRelationship value="org.picketlink.idm.model.Relationship" />
          <supportAllFeatures />
        </file>
      </stores>
    </named>

  </PicketLinkIDM>

</PicketLink>
```

You can take a look at *testsuite*[14] to see more examples.

Report a bug[15]

7.1.10.3. Bootstrap IDM from XML file

So to initialize Picketlink IDM from XML you can use the code like this:

```
// Replace with your own configuration file
String configFilePath = "config/embedded-file-config.xml";

ClassLoader tcl = Thread.currentThread().getContextClassLoader();
InputStream configStream = tcl.getResourceAsStream(configFilePath);
XMLConfigurationProvider xmlConfigurationProvider = new XMLConfigurationProvider();
IdentityConfigurationBuilder idmConfigBuilder =
  xmlConfigurationProvider.readIDMConfiguration(configStream);
```

Now you can bootstrap IDM from **`idmConfigBuilder`** in same way, like it's done in *Section 7.1.5, "Programmatic Configuration Overview"* . Note that you can initialize builder from XML file and then you can do some additional programmatic configuration. For example, you may need to programmatically add **`JPAContextInitializer`** in case that you are using JPA, because you will need access to JPA **`EntityManager`**.

Report a bug[16]

[14] https://github.com/picketlink/picketlink/tree/master/modules/idm/tests/src/test/resources/config
[15] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29723-626777+%5BLatest%5D&comment=Title%3A+XML+configuration+format%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29723-626777+01+Apr+2014+20%3A51+en-US+%5BLatest%5D
[16] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29724-620666+%5BLatest%5D&comment=Title%3A+Bootstrap+IDM+from+XML+file%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29724-620666+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

Identity Management - Working with JPA

8.1. JPAIdentityStoreConfiguration

The JPA identity store uses a relational database to store identity state. The configuration for this identity store provides control over which entity beans are used to store identity data, and how their fields should be used to store various identity-related state. The entity beans that store the identity data must be configured using the annotations found in the **`org.picketlink.jpa.annotations`** package. All identity configuration annotations listed in the tables below are from this package.

In the PicketLink site you can find useful information about how to provide your own custom identity model and use the annotations described in the next sections to store your custom types using the JPA Identity Store. Take a look at this guide for more details *http://picketlink.org/gettingstarted/custom_idm_model/*

Report a bug[1]

8.1.1. Default Database Schema

If you do not wish to provide your own JPA entities for storing IDM-related state, you may use the default schema provided by PicketLink in the **`picketlink-idm-simple-schema`** module. This module contains a collection of entity beans suitable for use with **`JPAIdentityStore`**. To use this module, add the following dependency to your Maven project's **`pom.xml`** file:

```
<dependency>
    <groupId>org.picketlink</groupId>
    <artifactId>picketlink-idm-simple-schema</artifactId>
    <version>${picketlink.version}</version>
</dependency>
```

In addition to including the above dependency, the default schema entity beans must be configured in your application's **`persistence.xml`** file. Add the following entries within the **`persistence-unit`** section:

```
<class>org.picketlink.idm.jpa.model.sample.simple.PartitionTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.AccountTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.RelationshipTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.AttributedTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.AttributeTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.AbstractCredentialTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.X509CredentialTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.GroupTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.OTPCredentialTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.TokenCredentialTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.RoleTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.PasswordCredentialTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.RelationshipIdentityTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.IdentityTypeEntity</class>
<class>org.picketlink.idm.jpa.model.sample.simple.DigestCredentialTypeEntity</class>
```

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29725-701176+%5BLatest%5D&comment=Title%3A+JPAIdentityStoreConfiguration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
<class>org.picketlink.idm.jpa.model.sample.simple.PermissionTypeEntity</class>
```

Report a bug[2]

8.1.2. Configuring an EntityManager

Before the JPA identity store can be used, it must be provided with an **`EntityManager`** so that it can connect to a database. In Java EE this can be done by providing a producer method within your application as follows:

```
@Produces
@PersistenceContext
private EntityManager picketLinkEntityManager;
```

If you are using multiple persistence units, you can specify which **`EntityManager`** should be used by PicketLink by specifying the **`@org.picketlink.annotations.PicketLink`** qualifier, for example like so:

```
@Produces
@PicketLink
@PersistenceContext
private EntityManager picketLinkEntityManager(unitName = "picketlink");
```

In the latter case, the **`EntityManager`** with the **`@PicketLink`** qualifier will always be preferred over any other **`EntityManager`** produced by your application.

Another important thing you should keep in mind when configuring the **`EntityManager`** is how you define the entity classes in your **`/META-INF/persistence.xml`**. By default, PicketLink will read all entities from your persistence unit to automatically discover those annotated with the PicketLink IDM JPA Annotations.

For example, for a simple application using the JPA Identity Store in conjunction with the Basic Identity Model you should have something like this:

```
<persistence-unit name="picketlink-forge-app-persistence-unit" transaction-type="JTA">
    <description>PicketLink IDM Persistence Unit Using the Basic Identity Model</description>

    // connection properties

    <class>org.picketlink.idm.jpa.model.sample.simple.PartitionTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.AccountTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.RelationshipTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.AttributedTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.AttributeTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.AbstractCredentialTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.X509CredentialTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.GroupTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.OTPCredentialTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.TokenCredentialTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.RoleTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.PasswordCredentialTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.RelationshipIdentityTypeEntity</class>
    <class>org.picketlink.idm.jpa.model.sample.simple.IdentityTypeEntity</class>
```

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29726-714911+%5BLatest%5D&comment=Title%3A+Default+Database+Schema%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29726-714911+03+Oct+2014+09%3A25+en-US+%5BLatest%5D

```
    <class>org.picketlink.idm.jpa.model.sample.simple.DigestCredentialTypeEntity</class>
</persistence-unit>
```

Report a bug[3]

8.1.3. Mapping IdentityType Types

The following table summarizes all annotations that can be used to map entities to **IdentityType** types:

Table 8.1. **IdentityType** Annotations

Annotation	Description	Property Type	Required
@IdentityManaged	This annotation is a type-level annotation and must be used to specify the **IdentityType** types that should be mapped by the annotated entity.	-	True
@AttributeValue	This annotation can be used to map a entity property to a **IdentityType** property. The **name** property of this annotation can be used in case the property names are different.	Any Type	False
@Identifier	The unique identifier value for the identity.	String	True
@IdentityClass	The type for the identity. When a **IdentityType** is stored the FQN of its type is stored in a property annotated with this annotation.	String	True
@OwnerReference	The reference to a **Partition** mapped entity. This annotation is used to identify the property that holds a reference to the partition where a **IdentityType** belongs. Usually this	The same type used to map a **Partition**	True

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29727-701175+%5BLatest%5D&comment=Title%3A+Configuring+an+EntityManager%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29727-701175+22+Aug+2014+02%3A39+en-US+%5BLatest%5D

Annotation	Description	Property Type	Required
	annotation is used in conjunction with a @ManyToOne property referencing the entity used to store partitions.		

The following code shows an example of an entity class configured to store **`User`** types:

Example 8.1. Example

```
@IdentityManaged (User.class)
@Entity
public class IdentityTypeEntity implements Serializable {

  @Id
  @Identifier
  private String id;

  @IdentityClass
  private String typeName;

  @AttributeValue
  private String loginName;

  @AttributeValue
  private Date createdDate;

  @AttributeValue
  private Date expirationDate;

  @AttributeValue
  private boolean enabled;

  @OwnerReference
  @ManyToOne
  private PartitionTypeEntity partition;

    // getters and setters
}
```

Report a bug[4]

8.1.4. Mapping `Partition` Types

The following table summarizes all annotations that can be used to map entities to **`IdentityType`** types:

Table 8.2. **`Partition`** Annotations

Annotation	Description	Property Type	Required
`@IdentityManaged`	This annotation is a type-level annotation	-	True

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29728-620667+%5BLatest%5D&comment=Title%3A+Mapping+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EIdentityType%3C%2Fcode%3E+Types%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29728-620667+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

Annotation	Description	Property Type	Required
	and must be used to specify the **`Partition`** types that should be mapped by the annotated entity.		
`@AttributeValue`	This annotation can be used to map a entity property to a **`Partition`** property. The **`name`** property of this annotation can be used in case the property names are different.	Any Type	False
`@Identifier`	The unique identifier value for the partition.	String	True
`@PartitionClass`	The type for the partition. When a **`Partition`** is stored the FQN of its type is stored in a property annotated with this annotation.	String	True
`@ConfigurationName`	This annotation must be used to indicate the field to store the configuration name for a partition.	String	True

The following code shows an example of an entity class configured to store **`Realm`** types:

Example 8.2. Example

```
@IdentityManaged (Realm.class)
@Entity
public class PartitionTypeEntity implements Serializable {

  @Id
  @Identifier
  private String id;

  @AttributeValue
  private String name;

  @PartitionClass
  private String typeName;

  @ConfigurationName
  private String configurationName;

    // getters and setters
}
```

Report a bug[5]

8.1.5. Mapping `Relationship` Types

The following table summarizes all annotations that can be used to map entities to **`Relationship`** types:

Table 8.3. **`Relationship`** Annotations

Annotation	Description	Property Type	Required
`@IdentityManaged`	This annotation is a type-level annotation and must be used to specify the **`Relationship`** types that should be mapped by the annotated entity.	-	True
`@AttributeValue`	This annotation can be used to map a entity property to a **`Relationship`** property. The **`name`** property of this annotation can be used in case the property names are different.	Any Type	False
`@Identifier`	The unique identifier value for the relationship.	String	True
`@RelationshipClass`	The type for the relationship. When a **`Relationship`** is stored the FQN of its type is stored in a property annotated with this annotation.	String	True
`@RelationshipDescriptor`	This annotation must be used to indicate the field to store the name of the relationship role of a member.	String	True
`@RelationshipMember`	The reference to a **`IdentityType`** mapped entity. This annotation is used to identify the property	The same type used to map a **`IdentityType`**	True

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29729-620668+%5BLatest%5D&comment=Title%3A+Mapping+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EPartition%3C%2Fcode%3E+Types%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29729-620668+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

Annotation	Description	Property Type	Required
	that holds a reference to the identity type that belongs to this relationship with a specific descriptor. Usually this annotation is used in conjunction with a @ManyToOne property referencing the entity used to store identity types.		
`@OwnerReference`	The reference to a **`Relationship`** mapped entity. This annotation is used to identify the property that holds a reference to the root entity for relationships, usually the entity annotated with the **`@RelationshipClass`** annotation.	The same type used to map an entity with the **`@RelationshipClass`** annotation.	True

The following code shows an example of an entity class configured to store **`Relationship`** types:

Example 8.3. Example

```
@IdentityManaged (Relationship.class)
@Entity
public class RelationshipTypeEntity implements Serializable {

  @Id
  @Identifier
  private String id;

  @RelationshipClass
  private String typeName;

    // getters and setters
}
```

When mapping a relationship you also need to provide a specific entity to store its members:

Example 8.4. Example

```

@Entity
public class RelationshipIdentityTypeEntity implements Serializable {

  @Id
  @GeneratedValue
  private Long id;

  @RelationshipDescriptor
```

```
  private String descriptor;

  @RelationshipMember
  @ManyToOne
  private IdentityTypeEntity identityType;

  @OwnerReference
  @ManyToOne
  private RelationshipTypeEntity owner;

    // getters and setters
}
```

Report a bug[6]

8.1.6. Mapping Attributes for `AttributedType` Types

The following table summarizes all annotations that can be used to map attributes to **`AttributedType`** types:

Table 8.4. **`Partition`** Annotations

Annotation	Description	Property Type	Required
`@AttributeName`	The name of the attribute. A property with this annotation is used to store the name of the attribute.	String	True
`@AttributeValue`	The value of the attribute. A property with this annotation is used to store the value of the attribute. Values are Base64 encoded.	String	True
`@AttributeClass`	The type for the attribute. When a attribute is stored the FQN of its type is stored in a property annotated with this annotation.	String	True
`@OwnerReference`	The reference to a **`IdentityType`**, or a **`Partition`** or a **`Relationship`** mapped entity. This annotation is used to identify the property	The same type used to map **`IdentityType`**, or a **`Partition`** or a **`Relationship`** type.	True

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29730-620669+%5BLatest%5D&comment=Title%3A+Mapping+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3ERelationship%3C%2Fcode%3E+Types%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29730-620669+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

Annotation	Description	Property Type	Required
	that holds a reference to the owner of the attributes.		

The following code shows an example of an entity class configured to store attributes for **`IdentityType`** types:

Example 8.5. Example

```
@Entity
public class IdentityTypeAttributeEntity implements Serializable {

  @OwnerReference
  @ManyToOne
  private IdentityTypeEntity owner;

  @AttributeClass
  private String typeName;

  @AttributeName
  private String name;

  @AttributeValue
  private String value;

    // getters and setters
}
```

Report a bug[7]

8.1.7. Mapping a `CredentialStorage` type

The following table summarizes all annotations that can be used to map attributes to **`AttributedType`** types:

Table 8.5. **`Partition`** Annotations

Annotation	Description	Property Type	Required
`@CredentialClass`	The type for the credential. When a credential is stored the FQN of its corresponding **`CredentialStorage`** type is stored in a property annotated with this annotation.	String	True

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29731-620670+%5BLatest%5D&comment=Title%3A+Mapping+Attributes+for+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EAttributedType%3C%2Fcode%3E+Types%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29731-620670+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

Annotation	Description	Property Type	Required
`@CredentialProperty`	This annotation can be used to map a entity property to a **`CredentialStorage`** property. The **`name`** property of this annotation can be used in case the property names are different.	String	True
`@EffectiveDate`	The effective date for a credential. A property annotated with this annotation will be mapped to the **`effectiveDate`** of a **`CredentialStorage`** type.	String	True
`@ExpiryDate`	The expiry date for a credential. A property annotated with this annotation will be mapped to the **`expiryDate`** of a **`CredentialStorage`** type.	String	True
`@OwnerReference`	The reference to a **`IdentityType`** mapped entity. This annotation is used to identify the property that holds a reference to the owner of the credential.	The same type used to map a **`IdentityType`** type.	True

The following code shows an example of an entity class configured to store password-based credentials for **`IdentityType`** types:

Example 8.6. Example

```
@ManagedCredential (EncodedPasswordStorage.class)
@Entity
public class PasswordCredentialTypeEntity implements Serializable {

  @Id
  @GeneratedValue
  private Long id;

  @OwnerReference
  @ManyToOne
  private IdentityTypeEntity owner;

  @CredentialClass
  private String typeName;
```

```
  @EffectiveDate
  private Date effectiveDate;

  @ExpiryDate
  private Date expiryDate;

  @CredentialProperty (name = "encodedHash")
  private String passwordEncodedHash;

  @CredentialProperty (name = "salt")
  private String passwordSalt;

    // getters and setters
}
```

Report a bug[8]

8.1.8. Configuring the Mapped Entities

Once your entities are properly mapped, you're ready to configure the JPA store with them. To do that you only need to:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
  .stores()
    .jpa()
      .mappedEntity(IdentityTypeEntity.class, PartitionTypeEntity.class, ...);
```

Report a bug[9]

8.1.9. Providing a `EntityManager`

Sometimes you may need to configure how the **`EntityManager`** is provided to the **`JPAIdentityStore`**, like when your application is using CDI and you must run the operations in the scope of the current transaction by using a injected **`EntityManager`** instance.

In cases like that, you need to initialize the **`IdentityContext`** by providing a **`ContextInitializer`** implementation, as discussed in *Section 7.1.9, "Identity Context Configuration"* . You can always provide your own implementation for this interface to obtain the **`EntityManager`** from your application's environment.

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
  .stores()
```

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29732-620671+%5BLatest%5D&comment=Title%3A+Mapping+a+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3ECredentialStorage%3C%2Fcode%3E+type%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29732-620671+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29733-620339+%5BLatest%5D&comment=Title%3A+Configuring+the+Mapped+Entities%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29733-620339+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

```
    .jpa()
      .addContextInitializer(new ContextInitializer() {
        @Override
        public void initContextForStore(IdentityContext context, IdentityStore<?> store) {
          if (store instanceof JPAIdentityStore) {
              EntityManager entityManager = // get the EntityManager
              context.setParameter(JPAIdentityStore.INVOCATION_CTX_ENTITY_MANAGER,
entityManager);
          }
        }
      });
```

In most cases you don't need to provide your own **`ContextInitializer`**.

Report a bug[10]

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29734-710508+%5BLatest%5D&comment=Title%3A+Providing+a+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EEntityManager%3C%2Fcode%3E%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29734-710508+17+Sep+2014+22%3A25+en-US+%5BLatest%5D

Identity Management - Working with LDAP

9.1. Overview

The LDAP Identity Store allows a LDAP Directory to be used as a source of identity data. Most organizations rely on a LDAP Directory to store users, groups, roles and relationships between those entities. Some of them only store users and groups, others only users and so forth. The point is that each organization has its own structure, how data is organized on the server and policies to govern all that. That said, is very hard to get all different use cases satisfied given all those nuances.

To try to overcome that, the LDAP Identity Store provides a simple and easy mapping between the entries in your LDAP tree and the PicketLink types (**`IdentityType`**, **`Relationship`** and so forth), plus some additional configuration options that give you more control how the store should integrate with your server.

The store can be used in read-only or read-write mode. Depending on your permissions on the server, you should consider one of these alternatives, otherwise you can get errors when, for example, trying to add, update or remove entries from the server.

The list below summarizes some of the most important capabilities provided by this store:

- Mapping **`IdentityType`** types to their corresponding LDAP entries and attributes.
- Mapping **`Relationship`** types to their corresponding LDAP entries and attributes.
- Mapping of parent/child relationships between the LDAP entries mapped to the same type.
- Authentication of users based on username/password credentials.
- Use of LDAP UUID attributes as the identifier for identity types. For each identity type in PicketLink we need to provide a single/unique identifier. The LDAP store uses the **`entryUUID`** and **`objectGUID`** (depending on your server implementation, of course) to identify each type. You can also specify a different attribute name if your LDAP server does not support any of these none attributes.

But the LDAP Directory has also some limitations (schema limitations, restrictive usage policies) and because of that the LDAP Identity Store does not supports all the feature set provided by PicketLink. The table below lists what is not supported by the LDAP Identity Store:

- *Chapter 6, Identity Management - Attribute Management*
- Complex relationship mappings such as **`GroupRole`**.
- Relationships can not be updated directly using the **`IdentityManager`**.
- Limited support for credential types. Only username/password is available.
- Partition Management.
- Permission Management.

Report a bug[1]

9.2. Configuration

The LDAP Identity Store can be configured as follows:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
    .named("ldap.config")
        .stores()
            .ldap()
                // connection configuration
                .baseDN("dc=jboss,dc=org")
                .bindDN("uid=admin,ou=system")
                .bindCredential("passwd")
                .url("ldap://localhost:389")

                // mapping configuration
                .mapping(Agent.class)
                    .baseDN("ou=Agent,dc=jboss,dc=org")
                    .objectClasses("account")
                    .attribute("loginName", "uid", true)
                    .readOnlyAttribute("createdDate", "createTimeStamp")
                .mapping(User.class)
                    .baseDN("ou=User,dc=jboss,dc=org")
                    .objectClasses("inetOrgPerson", "organizationalPerson")
                    .attribute("loginName", "uid", true)
                    .attribute("firstName", "cn")
                    .attribute("lastName", "sn")
                    .attribute("email", EMAIL)
                    .readOnlyAttribute("createdDate", "createTimeStamp")
                .mapping(Role.class)
                    .baseDN("ou=Roles,dc=jboss,dc=org")
                    .objectClasses("role")
                    .attribute("name", "cn", true)
                    .readOnlyAttribute("createdDate", "createTimeStamp")
                .mapping(Group.class)
                  .hierarchySearchDepth(4)
                  .objectClasses("group")
                  .attribute("name", "cn", true)
                  .readOnlyAttribute("createdDate", "createTimeStamp")
                  .parentMembershipAttributeName("member")
                .mapping(Grant.class)
                    .forMapping(Role.class)
                    .attribute("assignee", "member")
                .mapping(GroupMembership.class)
                    .forMapping(Group.class)
                    .attribute("member", "member");
```

Report a bug[2]

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29735-714898+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29735-714898+03+Oct+2014+07%3A27+en-US+%5BLatest%5D

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29736-620342+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

9.2.1. Connecting to the LDAP Server

The connection to your LDAP server can be configured as follows:

```
.ldap()
  .baseDN("dc=jboss,dc=org")
  .bindDN("uid=admin,ou=system")
  .bindCredential("passwd")
  .url("ldap://localhost:389")
```

You can also provide additional connection **`Properties`** that will be used when creating the **`LdapContext`**.

```
.ldap()
  .connectionProperties(myProperties)
```

The table below describes each configuration option:

Table 9.1. LDAP Connection Configuration Options

Option	Description
`baseDN`	Sets the base DN for a specific mapped type or all types.
`bindDN`	Sets the the DN used to bind against the ldap server. If you want to perform write operations the DN must have permissions on the agent,user,role and group contexts.
`bindCredential`	Sets the password for the bindDN.
`url`	Sets the url that should be used to connect to the server. Eg.: ldap://<<server>>:389 .
`connectionProperties`	Set a **`Properties`** instance from where additional connection properties will be retrieved from when creating the **`LdapContext`**.

Report a bug[3]

9.2.1.1. Connection Pooling

When working with a LDAP server to query, create/update/remove entries, one thing you should keep in mind is enabling connection pooling. Otherwise you may run into troubles in multi-threaded or load testing environments.

Connection pooling can be easily enabled by setting some few connection properties.

```
Properties properties = new Properties();

// set this property to enable connection pooling
properties.put("com.sun.jndi.ldap.connect.pool", "true");
```

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29737-620343+%5BLatest%5D&comment=Title%3A+Connecting+to+the+LDAP+Server%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
// provide other pooling properties to configure the pool
System.setProperty("com.sun.jndi.ldap.connect.pool.authentication", "simple");
System.setProperty("com.sun.jndi.ldap.connect.pool.maxsize", "10");
System.setProperty("com.sun.jndi.ldap.connect.pool.prefsize", "5");
System.setProperty("com.sun.jndi.ldap.connect.pool.timeout", "300000");
System.setProperty("com.sun.jndi.ldap.connect.pool.debug", "all");

IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

  builder
    .named(SIMPLE_LDAP_STORE_CONFIG)
        .stores()
            .ldap()
                .connectionProperties(properties) // set the connection properties to the
 LDAP configuration
                .baseDN(embeddedServer.getBaseDn())
                .bindDN(embeddedServer.getBindDn())
                .bindCredential(embeddedServer.getBindCredential())
                .url(embeddedServer.getConnectionUrl())
```

Note

The LDAP Identity Store relies on the Java SDK to provide connection pooling. For more details, please take a look at *http://docs.oracle.com/javase/jndi/tutorial/ldap/connect/config.html*.

Report a bug[4]

9.2.2. Mapping Identity Types

The LDAP configuration provides a simple mapping between your identity types and their corresponding LDAP entries. The way you map your types have a huge impact on how the LDAP Identity Store performs its operations.

Usually, a mapping is done as follows:

```
IdentityConfigurationBuilder builder = new IdentityConfigurationBuilder();

builder
    .named("ldap.config")
        .stores()
            .ldap()
                .mapping(User.class)
                    .baseDN("ou=User,dc=jboss,dc=org")
                    .objectClasses("inetOrgPerson", "organizationalPerson")
                    .attribute("loginName", "uid", true)
                    .attribute("firstName", "cn")
                    .attribute("lastName", "sn")
                    .attribute("email", "mail")
                    .readOnlyAttribute("createdDate", "createTimeStamp")
```

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30405-628696+%5BLatest%5D&comment=Title%3A+Connection+Pooling%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30405-628696+10+Apr+2014+06%3A52+en-US+%5BLatest%5D

For each mapping you need to provide the identity type being mapped (in the case above the **`User`** type) plus all information required to store the type and populate its properties from their corresponding LDAP attributes.

In the example above, we're considering that **`User`** entries are located at the *baseDN* "ou=User,dc=jboss,dc=org". The baseDN is a very important information, specially if you want to store information from a type instance. Beside that, the baseDN can have a huge impact on performance when querying your LDAP entries for a specific type, as the search will be more restrictive and consider only those entries located at the baseDN and sub entries.

Another important configuration is the *objectClass* list related with a type. The objectClass is very important when storing new entries in your LDAP server. Also, the objectClass helps the LDAP Identity Store to make better queries against your server by restricting which entries should be considered during the search based on the objectClass list you provide.

In order to store and retrieve attributes from the LDAP server, you need to map them to the properties of your type. The attribute mapping is pretty simple, you just provide the name of the property being mapped and its corresponding LDAP attribute name. An important aspect when mapping the attributes is that you should always configure an attribute as the identifier. In the example above, we're telling the LDAP configuration to consider the following attribute as an identifier:

```
.mapping(User.class)
  .attribute("loginName", "uid", true)
```

Report a bug[5]

9.2.3. Mapping Relationship Types

As mentioned before, the relationship support of the LDAP Identity Store is limited. But you can always map the most common relationships such as **`Grant`** and **`GroupMembership`**

```
.ldap()
  .mapping(Grant.class)
    .forMapping(Role.class)
      .attribute("assignee", "member"))
```

When mapping a relationship type you need to configure which identity type is the owner of a relationship. For example, when mapping a **`Grant`** relationship, the LDAP attribute used to map the association between a role and other types is the *member* attribute. This attribute belongs to role entries on the LDAP server, what makes the **`Role`** type the owner of this relationship. For last, we need to tell which property on the **`Grant`** type is related with the associated entries. In the case of the **`Grant`** relationship, we're configuring the *assignee* property to store the associated type instances.

Report a bug[6]

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29738-620344+%5BLatest%5D&comment=Title%3A+Mapping+Identity+Types%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29738-620344+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29739-620345+%5BLatest%5D&comment=Title%3A+Mapping+Relationship+Types%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29739-620345+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

9.2.4. Mapping a Type Hierarchies

The LDAP configuration supports the mapping of simple hierarchies (parent/child) of a single type. This is specially useful when mapping groups, for example. Where groups can have a parent and also child groups.

```
.ldap()
  .mapping(Group.class)
    .parentMembershipAttributeName("member")
```

In the example above, we're using the *member* attribute from LDAP to store the childs of a parent group.

In some cases, the performance can be impacted when retrieving parent/child hierarchies from the LDAP server. By default, the LDAP Identity Store is configure to resolve only three levels of hierarchies. But you can always override this configuration as follows:

```
.ldap()
  .mapping(Group.class)
    .hierarchySearchDepth(1)
```

In the example above, we're telling the LDAP Identity Store to consider only one level depth. Which means that only the direct parent of a group will be resolved.

Report a bug[7]

9.2.5. Mapping Groups to different contexts

Sometimes may be useful to map a specific group to a specific context or DN.

The following configuration maps the group with path */QA Group* to *ou=QA,ou=Groups,dc=jboss,dc=org*

```
mapping(Group.class)
  .baseDN(embeddedServer.getGroupDnSuffix())
  .objectClasses(GROUP_OF_NAMES)
  .attribute("name", CN, true)
  .readOnlyAttribute("createdDate", CREATE_TIMESTAMP)
  .parentMembershipAttributeName("member")
  .parentMapping("QA Group", "ou=QA,ou=Groups,dc=jboss,dc=org")
```

With this configuration you can have groups with the same name, but with different paths.

```
IdentityManager identityManager = getIdentityManager();
Group managers = new SimpleGroup("managers");

identityManager.add(managers); // group's path is /manager

Group qaGroup = identityManager.getGroup("QA Group");
Group managersQA = new SimpleGroup("managers", qaGroup);
```

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29740-620346+%5BLatest%5D&comment=Title%3A+Mapping+a+Type+Hierarchies%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29740-620346+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

```
// the QA Group is mapped to a different DN.
Group qaManagerGroup = identityManager.add(managersQA); // group's path is /QA Group/managers
```

Report a bug[8]

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29741-620347+%5BLatest%5D&comment=Title%3A+Mapping+Groups+to+different+contexts%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29741-620347+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

Identity Management - Permissions API and Permission Management

10.1. Overview

The Permissions API is a set of extensible authorization features that provide capabilities for determining access privileges for application resources. This chapter describes the ACL (Access Control List) features and the management of persistent resource permissions via the **`PermissionManager`**. It also explains how the **`PermissionResolver`** SPI can be used to in conjunction with a custom **`PermissionVoter`** implementation, allowing you to plugin your own custom authorization logic.

The **`Permission`** interface is used in a number of places throughout the Permissions API, and defines the following methods:

```
public interface Permission {

    Object getResource();

    Class<?> getResourceClass();

    Serializable getResourceIdentifier();

    IdentityType getAssignee();

    String getOperation();
}
```

Each permission instance represents a specific resource permission, and contains three important pieces of state:

- The *assignee*, which is the identity to which the permission is assigned.
- The *operation*, which is a string value that represents the exact action that the assigned identity is allowed to perform.
- Either a direct reference to the *resource* (if known), or a combination of a *resource class* and *resource identifier*. This value represents the resource to which the permission applies.

To understand better what a permission is for, here are some examples:

- John is allowed to read FileA.txt
- John is allowed to load/read entity Confidential with identifier 123.
- John is allowed to view the /myapp/page page.
- John is allowed to access Mary's profile
- John is allowed to view button 'Delete' on page /myapp/page.

Basically, permissions can be **`string`** or **`type-based`**. In the latter case, if you are using JPA entities, you can also manage permissions for a specific entity given its identifier.

Permissions can be also grouped into roles or groups. Or any other **`IdentityType`** you want. For example, let's say that we have a role wich gives read access to a file. If you grant this role to users

they are going to inherit all privileges/permissions from the role they were granted. You can even grant this role to a entire group, where all members of the group are also going to inherit the privileges.

Report a bug[1]

10.2. Checking permissions for the current user

The primary method for accessing the Permissions API is via the **`Identity`** bean, which provides the following two methods for checking permissions for the currently authenticated user:

```
boolean hasPermission(Object resource, String operation);
boolean hasPermission(Class<?> resourceClass, Serializable identifier, String operation);
```

The first overloaded method may be used when you have a reference to the actual resource for which you wish to check privileges:

```
@Inject Identity identity;

public void deleteAccount(Account account) {
    // Check the current user has permission to delete the account
    if (identity.hasPermission(account, "DELETE")) {
        // Logic to delete Account object goes here
    } else {
        throw new SecurityException("Insufficient privileges!");
    }
}
```

The second overloaded method may be used when you *don't* have a reference to the resource object, but you have it's *identifier* value (for example the primary key value of an entity bean):

```
@Inject Identity identity;

public void deleteCustomer(Long customerId) {
    // Check the current user has permission to delete the customer
    if (identity.hasPermission(Customer.class, customerId, "DELETE")) {
        // Logic to delete Customer object goes here
    } else {
        throw new SecurityException("Insufficient privileges!");
    }
}
```

This method is generally used for performance reasons, for example when you don't necessarily wish to load a resource object because of a possibly expensive instantiation cost, or you wish to check whether there are suitable permissions assigned to the user before loading the resource.

Report a bug[2]

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29742-713897+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29742-713897+30+Sep+2014+01%3A09+en-US+%5BLatest%5D

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29743-620349+%5BLatest%5D&comment=Title%3A+Checking+permissions+for+the+current+user%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29743-620349+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

10.3. ACL Permissions

An ACL (Access Control List) can be used to control which identities may invoke specific operations on application resources. Underneath the covers, ACL security checks are handled by the **`PersistentPermissionResolver`**, which reads the ACL entries for each resource via the **`PermissionStore`**, which is typically a wrapper around some form of persistent storage such as database table.

Report a bug[3]

10.3.1. The PermissionManager Bean

ACL permissions are managed via the **`PermissionManager`**. Actually, this bean is used by the **`Identity`** when you invoke its **`hasPermission`** methods. An instance of this bean can be obtained by injecting the **`PermissionManager`** as follows:

```
@Inject PermissionManager permissionManager;
```

Note

The **`PermissionManager`** bean is injected based on the current partition defined by the application. If no partition was specified, the bean is injected based on the default partition.

You can also obtain a reference to the **`PermissionManager`** from a **`PartitionManager`** via the **`createPermissionManager()`** method:

```
@Inject PartitionManager partitionManager;

public void managePermissions() {
    PermissionManager permissionManager = partitionManager.createPermissionManager();
}
```

In the example above, the **`PermissionManager`** is being created based on the default partition. But you may also specify a partition when creating it:

```
@Inject PartitionManager partitionManager;

public void managePermissions() {
    Realm partition = partitionManager.getPartition(Realm.class, "MyPartition");
    PermissionManager permissionManager =
 partitionManager.createPermissionManager(partition);
}
```

Once you have a reference to the **`PermissionManager`**, you can use it to grant permissions:

```
public void allowRead(User user, Customer customer) {
```

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29744-620350+%5BLatest%5D&comment=Title%3A+ACL+Permissions%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
    permissionManager.grantPermission(user, customer, "READ");
}
```

The **`grantPermission()`** method accepts three parameters:

```
void grantPermission(IdentityType assignee, Object resource, String operation);
```

The *assignee* is the identity to which you wish to grant the permission. The *resource* is the application resource for which the permission applies. The *operation* is a String value representing the action that the assignee may invoke in relation to the resource.

Resources may conceivably be any type of **`Object`** so long as there exists a unique, serializable value that can be determined or in some way calculated from the resource object, which uniquely identifies that resource from other resources of the same type. This unique value is called the *identifier*, an example of which might be the primary key value of an entity bean. The **`PermissionHandler`** SPI (see section below) is responsible for generating identifier values for resource objects.

The **`revokePermission()`** method is used to remove permissions. Like **`grantPermission()`**, it also accepts three parameters:

```
void revokePermission(IdentityType assignee, Object resource, String operation);
```

It is also possible to revoke all assigned permissions for a single resource via the **`clearPermissions()`** method. This is useful for example if you wish to delete the resource and don't wish to leave orphaned permissions:

```
void clearPermissions(Object resource);
```

There are also a number of overloaded methods available for querying permissions. These methods take an assortment of parameters depending on exactly which permissions you wish to find:

```
List<Permission> listPermissions(Object resource);

List<Permission> listPermissions(Class<?> resourceClass, Serializable identifier);

List<Permission> listPermissions(Object resource, String operation);

List<Permission> listPermissions(Class<?> resourceClass, Serializable identifier,
                                 String operation);
```

Here's some examples:

```
// List all permissions for a known Product
Product p = lookupProduct("grapes");
List<Permission> permissions = permissionManager.listPermissions(p);

// List all permissions for a Product where we know the resource class
// and the identifier
List<Permission> permissions = permissionManager.listPermissions(
    Product.class, "bananas");

// List all "DELETE" permissions that have been granted for a Product
Product p = lookupProduct("apples");
List<Permissions> permissions = permissionManager.listPermissions(p, "DELETE");

// List all "UPDATE" permissions for a Product where we know the
// resource class and the identifier
```

```
List<Permissions> permissions = permissionManager.listPermissions(
    Product.class, "oranges", "UPDATE");
```

Report a bug[4]

10.3.2. Configuring resources for ACL usage

Every resource class for which you wish to support ACL permissions is required to have a corresponding **`PermissionHandler`**. This interface is primarily responsible for the generation of resource identifier values, plus a couple of other utility methods (please refer to the API documentation for more details):

```
public interface PermissionHandler {

    boolean canHandle(Class<?> resourceClass);

    Serializable getIdentifier(Object resource);

    Class<?> unwrapResourceClass(Object resource);

    Set<String> listClassOperations(Class<?> resourceClass);

    Set<String> listInstanceOperations(Class<?> resourceClass);
}
```

There are two ways that a resource class can be associated with a **`PermissionHandler`** - the *first* way is by providing a **`@PermissionsHandledBy`** annotation on the resource class itself:

```
import org.picketlink.idm.permission.annotations.PermissionsHandledBy;

@PermissionsHandledBy(CustomPermissionHandler.class)
public class MyResourceClass {

}
```

For the circumstances where it is not possible to annotate the resource class directly, the second way is to register a custom **`PermissionHandler`** instance for which the **`canHandle()`** method returns **`true`** for the resource class:

```
public boolean canHandle(Class<?> resourceClass) {
    return MyResourceClass.class.equals(resourceClass);
}
```

Registering a custom **`PermissionHandler`** is very easy - simply include it in your application deployment as an **`@ApplicationScoped`** bean, and it will be registered automatically. Here's a complete example of a **`PermissionHandler`** that allows permissions to be assigned to arbitrary string values (this handler is actually provided by PicketLink):

```
@ApplicationScoped
public class StringPermissionHandler implements PermissionHandler {
    @Override
```

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29745-713899+%5BLatest%5D&comment=Title%3A+The+PermissionManager+Bean%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29745-713899+30+Sep+2014+01%3A38+en-US+%5BLatest%5D

```
    public boolean canHandle(Class<?> resourceClass) {
        return String.class.equals(resourceClass);
    }

    @Override
    public Serializable getIdentifier(Object resource) {
        checkResourceValid(resource);
        return (String) resource;
    }

    @Override
    public Class<?> unwrapResourceClass(Object resource) {
        checkResourceValid(resource);
        return String.class;
    }

    private void checkResourceValid(Object resource) {
        if (!(resource instanceof String)) {
            throw new IllegalArgumentException("Resource [" + resource +
                "] must be instance of String");
        }
    }

    @Override
    public Set<String> listAvailableOperations(Class<?> resourceClass) {
        return Collections.emptySet();
    }
}
```

Report a bug[5]

10.3.3. Restricting resource operations

For many resource types it makes sense to restrict the set of resource operations for which permissions might be assigned. For example, an application might have an entity bean containing lookup values for countries. This **`Country`** bean is likely to only require the bare minimum in terms of data management and so you might like to restrict the available operations for it to the typical **`CREATE, READ, UPDATE, DELETE`** operations. To do this we use the **`@AllowedOperations`** annotation - this annotation allows us to provide a child array of **`@AllowedOperation`** values that specify exactly which operation values that permissions can be assigned for:

```
import org.picketlink.idm.permission.annotations.AllowedOperation;
import org.picketlink.idm.permission.annotations.AllowedOperations;

@Entity
@AllowedOperations({
    @AllowedOperation(value = "CREATE", mask = 1, classOperation = true),
    @AllowedOperation(value = "READ", mask = 2),
    @AllowedOperation(value = "UPDATE", mask = 4),
    @AllowedOperation(value = "DELETE", mask = 8)
})
public class Country implements Serializable {
```

The optional **`mask`** value can be used to specify a bitmask value to allow for more efficient storage of permission values. If the mask values are set, the operation values for that object's permissions will be

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29746-620352+%5BLatest%5D&comment=Title%3A+Configuring+resources+for+ACL+usage%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29746-620352+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

stored as a numerical value with the corresponding bit values turned on. For example, if a single user was assigned permission for both the **READ** and **UPDATE** operations for one of our **Country** beans, then this operation value would be stored as 6 (**READ** (2) + **UPDATE** (4)).

The other optional value, **classOperation** can be set to **true** if the permission applies to the class itself, and not an instance of a class. For example, you might wish to check that the current user has permission to actually create a new **Country** bean. In this case, the permission check would look something like this:

```
@Inject Identity identity;

public void createCountry() {
    if (!identity.hasPermission(Country.class, "CREATE")) {
        throw new SecurityException(
          "Current user has insufficient privileges for this operation.");
    }

    // snip
}
```

This functionality is provided by the **ClassPermissionHandler** permission handler.

Report a bug[6]

10.4. PermissionResolver SPI

Report a bug[7]

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29747-620353+%5BLatest%5D&comment=Title%3A+Restricting+resource+operations%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29747-620353+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29748-620354+%5BLatest%5D&comment=Title%3A+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EPermissionResolver%3C%2Fcode%3E+SPI%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29748-620354+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

Authorization

11.1. Overview

Authorization is the act of specifying access rights and enforce them consistently when accessing protected resources. PicketLink offers a rich and extensible authorization API that allows for significant customization of the authorization process, while also providing sensible default authorization rules/policies and checks for developers that wish to get up and running quickly.

In conjunction with Identity Management, PicketLink provides an end-to-end authorization where you can easily specify access rights and also enforce them to your POJO, EJB or RESTFul endpoints, for example. This chapter will endeavour to describe the authorization API and the authorization process, becoming a good place to gain a general overall understanding of authorization in PicketLink. It is very important that you have some background about how Authentication and Identity Management works in PicketLink. For more detalis, check their respective chapters.

Report a bug[1]

11.2. Configuration

To enable authorization to your project you must provide the following configuration to your **`WEB-INF/beans.xml`** or **`META-INF/beans.xml`**:

```
<interceptors>
 <class>org.apache.deltaspike.security.impl.extension.SecurityInterceptor</class>
</interceptors>
```

The interceptor above is responsible to intercept all invocations to your CDI beans and check for authorization rules and policies before processing their methods.

Report a bug[2]

11.3. Role-Based Access Control

RBAC allows you to perform authorization based on the roles granted for an user. For that, PicketLink provides a specific annotation. You only need to specify the role name.

```
@RolesAllowed("Administrator")
public void shutdown() {
 // only users granted with this role can access this method
}
```

The **`@RolesAllowed`** annotation can also be used on types. In this case, all bean methods are protected:

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35787-665371+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35787-665371+07+Jun+2014+09%3A55+en-US+%5BLatest%5D

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35788-665373+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35788-665373+07+Jun+2014+10%3A05+en-US+%5BLatest%5D

```
@RolesAllowed("Administrator")
public class MyBean() {

}
```

You can also define multiple roles if you want to:

```
@RolesAllowed({"Sales", "Financial"})
```

Report a bug[3]

11.4. Group-Based Access Control

GBAC allows you to perform authorization based on the groups defined for an user. For that, PicketLink provides a specific annotation. You only need to specify the group name.

```
@GroupsAllowed("Managers")
public void approveTimesheet() {
 // only users form group "Project Manager" are allowed to access this method
}
```

The **`@GroupsAllowed`** annotation can also be used on types. In this case, all bean methods are protected:

```
@GroupsAllowed("Managers")
public class Timesheet() {

}
```

You can also define multiple groups if you want to:

```
@GroupsAllowed({"Marketing", "Human Resources"})
```

Report a bug[4]

11.5. Partition-Based Access Control

PicketLink uses *partitions* to logically separate identities such as user, roles and groups. A very common requirement for SaaS applications where multi-tenancy may be an important requirement.

The **`@PartitionsAllowed`** annotation allows you to restrict access for beans based on the partition an user belongs. You only need to specify the partition name.

```
@PartitionsAllowed("Acme Realm")
```

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35792-665388+%5BLatest%5D&comment=Title%3A+Role-Based+Access+Control%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35792-665388+07+Jun+2014+11%3A45+en-US+%5BLatest%5D

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35793-665389+%5BLatest%5D&comment=Title%3A+Group-Based+Access+Control%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35793-665389+07+Jun+2014+11%3A46+en-US+%5BLatest%5D

```
public void someMethod() {
 // only users from "Acme Realm" partition are allowed to access this method
}
```

The **@PartitionsAllowed** annotation can also be used on types. In this case, all bean methods are protected:

```
@PartitionsAllowed("Acme Realm")
public class ACMEServices() {

}
```

You can also define multiple partitions if you want to:

```
@PartitionsAllowed({"ACME Realm", "Foo Realm"})
```

Partitions in PicketLink are represented by a specific type. For example, by default PicketLink provides a **org.picketlink.idm.model.basic.Realm** type to represent them. Considering that PicketLink allows you to provide your own types, the **@PartitionsAllowed** can also be used to restrict which partition types are allowed to perform an operation:

```
@PartitionsAllowed(type = {Acme.class, Foo.class})
```

Where **Acme** and **Foo** are partition types, implementing the **org.picketlink.idm.model.Partition** interface.

You can even combine both configurations (name and type) if you want.

Report a bug[5]

11.6. Restricting Access Based on the Authenticated User

It is very common to restrict access to a resource based on the state of the current user. Basically, check whether he is authenticated or not.

The **@LoggedIn** annotation allows you to protect a bean and allow access only from previously authenticated users.

```
@LoggedIn
public void logout() {
 // only authenticated users can logout
}
```

The **@LoggedIn** annotation can also be used on types. In this case, all bean methods are protected:

```
@LoggedIn
public class MyRESTAPI() {

}
```

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35789-665390+%5BLatest%5D&comment=Title%3A+Partition-Based+Access+Control%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35789-665390+07+Jun+2014+11%3A48+en-US+%5BLatest%5D

Accounts in PicketLink are represented by a specific type. For example, by default PicketLink provides a **`org.picketlink.idm.model.basic.User`** type to represent them. Considering that PicketLink allows you to provide your own types, the **`@LoggedIn`** can also be used to restrict which account types are allowed to perform an operation:

```
@LoggedIn(requiresAccount = {Employee.class, Customer.class})
```

Where **`Employee`** and **`Customer`** are account types, implementing the **`org.picketlink.idm.model.Account`** interface.

Report a bug[6]

11.7. Checking for Permissions

The PicketLink Permission API functionality is extended with the **`@RequiresPermission`** annotation. You can easily restrict access to your beans and methods based on the permissions granted for an user given a specific resource and operation.

```
@RequiresPermission(resource = "user_profile", operation = "read")
public javax.ws.rs.core.Response getUserProfile() {

}
```

Report a bug[7]

11.8. Using EL-Based Expresions

As an alternative to the built-in authorization annotations, PicketLink also supports EL-Based Expressions to define authorization constraints.

```
@Restrict("#{identity.loggedIn}")
public void protectedFromUnauthenticatedUsers() {

}

@Restrict("#{isLoggedIn()}")
public void protectedFromUnauthenticatedUsersFunction() {

}

@Restrict("#{hasPermission('user_profile','read')}")
public void protectedWithResourcePermission() {

}

@Restrict("#{hasPermission('profile','write')}")
public void protectedWithResourceWithoutPermission() {
```

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35795-673555+%5BLatest%5D&comment=Title%3A+Restricting+Access+Based+on+the+Authenticated+User%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35795-673555+17+Jun+2014+01%3A00+en-US+%5BLatest%5D

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35790-665383+%5BLatest%5D&comment=Title%3A+Checking+for+Permissions%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35790-665383+07+Jun+2014+11%3A13+en-US+%5BLatest%5D

```
}

@Restrict("#{hasRole('Tester')}")
public void protectedWithRequiredRole() {

}

@Restrict("#{hasRole('Invalid Role')}")
public void protectedWithRequiredInvalidRole() {

}

@Restrict("#{isMember('QA')}")
public void protectedWithRequiredGroup() {

}

@Restrict("#{isMember('Invalid Group')}")
public void protectedWithRequiredInvalidGroup() {

}

@Restrict("#{isMember('QA') and hasRole('Tester')}")
public void protectedWithRequiredMemberAndRole() {

}

@Restrict("#{isMember('QA') and hasRole('Invalid Role')}")
public void protectedWithRequiredMemberAndInvalidRole() {

}

@Restrict("#{hasPartition('default')}")
public void protectedWithRequiredPartitionName() {

}

@Restrict("#{hasPartition('invalid partition')}")
public void protectedWithInvalidPartitionName() {

}

@Restrict("#{hasAttribute('someAttribute')}")
public void protectedWithAttribute() {

}

@Restrict("#{hasAttribute('invalidAttribute')}")
public void protectedWithInvalidAttribute() {

}

@Restrict("#{identity.account != null}")
public void protectedWithValidAccountExpression() {

}

@Restrict("#{identity.account.partition.name == 'default'}")
public void protectedWithValidPartitionExpression() {

}

@Restrict("#{identity.account.partition.name != 'default'}")
public void protectedWithInvalidPartitionExpression() {

}
```

```
@Restrict("#{identity.account.attributes['someAttribute'] != null}")
public void protectedWithValidAccountAttributeExpression() {

}

@Restrict("#{identity.account.attributes['someAttribute'] == 'someValue'}")
public void protectedWithValidAccountAttributeValueExpression() {

}

@Restrict("#{identity.account.attributes['someAttribute'] == 'invalidValue'}")
public void protectedWithInvalidAccountAttributeValueExpression() {

}
```

EL expressions leverage the authorization capabilities by providing access to some additional functions and information like:

- **`#{identity}`** - The current **`Identity`** bean instance representing the authenticated user. From there you can invoke all public methods defined by this interface.

  `#{hasAttributes('someAttribute')}` - A handy function that checks if the authenticated user is set with a specific ad-hoc attribute.

Report a bug[8]

11.9. Providing Your Own Security Annotations

You may find that the built-in annotations provided by PicketLink are not enough to represent a very specific use case or requirement of your project. In this case, you can always provide your own authorization annotations and logic.

PicketLink Authorization is based on Apache DeltaSpike Security, which means you can use its API and adapt PicketLink accordingly to your needs. For more details, please check the following documentation *http://deltaspike.apache.org/security.html*

Report a bug[9]

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35794-665384+%5BLatest%5D&comment=Title%3A+Using+EL-Based+Expresions%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35794-665384+07+Jun+2014+11%3A24+en-US+%5BLatest%5D

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+35791-665386+%5BLatest%5D&comment=Title%3A+Providing+Your+Own+Security+Annotations%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=35791-665386+07+Jun+2014+11%3A35+en-US+%5BLatest%5D

Http Security

12.1. Overview

PicketLink provides an easy and simple API to protect your web application resources (eg.: pages and RESTFul endpoints) based on the HTTP protocol and fully integrated with the Java Servlet API.

With PicketLink you can have an additional security layer to filter every single request and apply security policies accordingly for specific resources or paths in your application. The security policies are usually based on the two major areas of application security: authentication and authorization.

Additionally, PicketLink also provides support for the most common HTTP protection mechanisms and against vulnerabilities such as:

- *CORS*, Cross Origin Resource Sharing (In Development)
- *CSRF*, Cross-Site Request Forgery (In Development)

The configuration is pretty simple and only requires a few lines of code. You'll see in the next sections the configuration in more details, but a simple setup would look like:

```
public class HttpSecurityConfiguration {

    public void configureHttpSecurity(@Observes SecurityConfigurationEvent event) {
        SecurityConfigurationBuilder builder = event.getBuilder();

        builder
            .http()
                .allPaths()
                    .authenticateWith()
                        .form()
                            .loginPage("/login.html")
                            .errorPage("/loginFailed.html");
    }
}
```

The example above is enough to enforce authentication to all your application resources and pages using a FORM to ask your users for credentials.

Report a bug[1]

12.2. Configuration

The configuration is based on which web resources or paths you want to protect. Paths always start with a slash and are relative to your application's context path. Examples of paths are:

- */protected/**, defines all resources starting with /protected. eg.: /protected/admin
- */protected*, defines a single resource /protected
- */*.jsf*, defines all resources with a suffix .jsf

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41509-701270+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41509-701270+22+Aug+2014+04%3A06+en-US+%5BLatest%5D

- */home.jsf*, defines a single resource /home.jsf

Once you define which paths you want to protect, you just need to provide the security policies you want to enforce for each of them. The security policies are related with the two major areas of application security: authentication and authorization.

Let's suppose you want to enforce FORM authentication to all your JSF pages and define for some of them which roles are allowed to access.

```
public class HttpSecurityConfiguration {

    public void configureHttpSecurity(@Observes SecurityConfigurationEvent event) {
        SecurityConfigurationBuilder builder = event.getBuilder();

        builder
            .http()
                .forPath("/*.jsf")
                    .authenticateWith()
                        .form()
                            .loginPage("/login.jsf")
                            .errorPage("/loginFailed.jsf")
                .forPath("/admin/*")
                    .authorizeWith()
                        .role("Administrator");
    }
}
```

You may notice that in the configuration above we're defining two path configurations. One more generic, covering all JSF pages and enforcing FORM authentication. The second, */admin/**, is enforcing RBAC(Role-Based Access Control) for a subset of paths starting with a specific pattern.

In this case, PicketLink will enforce FORM authentication to all your JSF pages but only allow access to the */admin/** pages if the authenticated user is granted with a *Administrator* role.

Simple isn't it ? The most important thing you should know is that all configuration is defined using an easy and fluent API provided by **`org.picketlink.config.http.HttpSecurityBuilder`**, retrieved when you invoke the **`http()`** method of **`org.picketlink.config.SecurityConfigurationBuilder`**. And also, that you must observe for the **`SecurityConfigurationEvent`** in order to obtain a reference to the **`HttpSecurityBuilder`** and start providing your security policies.

Report a bug[2]

12.2.1. Protecting Paths

As mentioned before, all configuration is based on the web resources or paths you want to protect. The **`HttpSecurityBuilder`** provides you a few methods to start protecting the paths of your application.

The most common way to configure a path is using the following method:

```
httpBuilder.forPath("/*.jsf")
```

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A41514-701283+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

It expects a string value starting with a slash and relative to your application's context path. PicketLink will try to match a incoming request to a specific path configuration using the value you provided to this method. You can provide this value using any pattern, such as those supported by the Java Servlet API specification. Keep in mind that depending on the paths you provided PicketLink will decide which requests should be handled or not.

The path is the main identifier of a HTTP resource. But you may want to handle different requests to the same path based on additional information such as the request method or headers. By default, PicketLink will match any request to a particular path regardless these additional information. But, if you want to configure a path with a specific method or header you can do so.

Let's assume you have a path */user/profile*. And a configuration as follows:

```
httpBuilder
    .forPath("/user/profile")
        .authenticateWith()
            .form()
    .forPath("/user/profile")
        .withHeaders()
            .header("X-Requested-With", "XMLHttpRequest")
        .authenticateWith()
            basic()
```

This is a useful example about how to support different authentication mechanisms from a single path. In this case, PicketLink will identify the proper configuration for a specific path based on the existence of the *X-Requested-With* header. If this header is present, PicketLink will enforce BASIC authentication, otherwise FORM will be used.

The same applies for HTTP methods. You may distinguish requests for a specific path based on the method of a incoming request.

You may also tell PicketLink that a specific path **`should not`** be protected, for that you just need to:

```
httpBuilder
    .forPath("/unprotected/resource")
        .unprotected()
```

In this case, PicketLink will just allow any request to a specific resource without enforce any security policy.

The **`HttpSecurityBuilder`** also provides some additional methods to configure your paths, some of them we'll cover in more details on the next sections.

Report a bug[3]

12.2.2. Grouping Paths

When defining your security policies you may want to reuse a set of configurations between different paths. This is very handy if you want to keep your configuration simple and group your paths based on a specific behavior.

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41517-701317+%5BLatest%5D&comment=Title%3A+Protecting+Paths%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41517-701317+22+Aug+2014+04%3A13+en-US+%5BLatest%5D

The **`HttpSecurityBuilder`** provides the following method to define a group:

```
httpBuilder
    .forGroup("REST Services")
        .authenticateWith()
            .basic()
    .forGroup("Web Pages")
        .authenticateWith()
            .form()
    .forPath("/rest/*", "REST Services")
    .forPath("/*.jsf", "Web Pages")
```

This is a very simple configuration that defines two groups: **`REST Services`** and **`Web Pages`**. The first is enforcing BASIC authentication and the latter FORM authentication. For that we used the **`forGroup`** method provided by the **`HttpSecurityBuilder`**.

In order to specify which group a specific path belongs to, you just need to do the following:

```
httpBuilder
    .forPath("/rest/*", "REST Services")
    .forPath("/*.jsf", "Web Pages")
```

In this case, we use the **`forPath(String, String)`** method from the **`HttpSecurityBuilder`** where the first argument is the path and the second the name of the group.

You can even override the configuration defined by a group when configuring a path. Let's say you want to allow access for a specific path based on a specific role. But still reusing all the configuration defined by the group.

```
httpBuilder
    .forPath("/rest/admin/*", "REST Services")
        .authorizeWith()
            .role("Administrator")
```

Report a bug[4]

12.2.3. Path Rewriting

PicketLink allows you to rewrite paths based on the information from the authenticated user. This is very useful if you want to use dynamic paths based on user information.

Let's suppose you want to request a path that changes based on the user identifier.

```
httpBuilder
    .forPath("/user/profile/{identity.account.id}")
```

Here, if the authenticated user has an identifier with value *1*, PicketLink is going to translate the real path as follows:

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41515-701427+%5BLatest%5D&comment=Title%3A+Grouping+Paths%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41515-701427+22+Aug+2014+04%3A24+en-US+%5BLatest%5D

```
/user/profile/1
```

You can do the same thing regarding the partition an user belongs to. Let's suppose your paths are defined based on the partition the user belongs to.

```
httpBuilder
    .forPath("/company/{identity.account.patition.name}/home.jsf")
```

In thise case, considering that the authenticated user belongs to a partition with name *acme*, PicketLink is going to translate the real path to:

```
/company/acme/home.jsf
```

This can be very useful for multi-tenancy applications.

You can use whatever function or object supported by PicketLink EL. Take a look at *Section 11.8, "Using EL-Based Expresions"* for more details.

Report a bug[5]

12.2.4. Path Redirection

PicketLink allows you to define redirection rules for a specific path. This feature allows you to perform a HTTP redirect depending on a specific condition.

The most simple example about how to use this feature is when you defined a logout path to your application.

```
httpBuilder
    .forPath("/logout")
        .logout()
        .redirectTo("/goodbye.html");
```

In this case, once the user is logged out, PicketLink will redirect the user to a */goodbye.html* page.

But some times you may want to perform a redirect depending on a specific condition. For example, let's define a rule where if a request is not authorized the user will be redirect to a specific error page:

```
httpBuilder
    .forPath("/admin/*")
        .authorizeWith()
            .role("Administrator")
        .redirectTo("/accessDenied.html").whenForbidden();
```

You can do the same thing if some error occurs during the request processing.

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41530-701454+%5BLatest%5D&comment=Title%3A+Path+Rewriting%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41530-701454+22+Aug+2014+04%3A26+en-US+%5BLatest%5D

```
httpBuilder
    .forPath("/admin/*")
        .authorizeWith()
            .role("Administrator")
        .redirectTo("/error.html").whenError();
```

If you want to simplify your configuration, you can define all redirect rules by *Section 12.2.2, "Grouping Paths"*.

Report a bug[6]

12.2.5. Permissive vs Restrictive

By default PicketLink will not enforce security for paths not explicity defined in the configuration. This is what we call the **`permissive`** behavior. It helps to get a more simple configuration and avoids you to specify every single path you want to protect or not.

But PicketLink also allows you to change this behavior and enforce that all your paths, regardless if they were defined in the configuration or not, should be protected. In this case, you should specify all paths you want to protect and specially those you don't want to enforce security such as CSS, JS and images files. We call this behavior as **`restrictive`**.

To enable the **`restrictive`** behavior you just need to provide the following configuration:

```
httpBuilder
    .restrictive()
```

Report a bug[7]

12.3. Authentication

PicketLink supports different authentication schemes, they are:

- **`HTTP BASIC`**
- **`HTTP DIGEST`**
- **`HTTP X.509 or CLIENT-CERT`**
- **`FORM`**
- **`Token-Based`**
- **`Write Your Own Method`**

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41531-701477+%5BLatest%5D&comment=Title%3A+Path+Redirection%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41531-701477+22+Aug+2014+04%3A27+en-US+%5BLatest%5D

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41511-700800+%5BLatest%5D&comment=Title%3A+Permissive+vs+Restrictive%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41511-700800+21+Aug+2014+09%3A26+en-US+%5BLatest%5D

When you configure the authentication policies to a specific path you just need to provide any of the available authentication schemes available from the **`authenticateWith()`** method provided by the **`HttpSecurityBuilder`**.

```
httpBuilder
    .forPath("/rest/*")
        .authenticateWith()
            .basic()
```

Some authentication schemes provide additional configuration to configure a specific behavior. In the next sections we'll cover each of them in more details.

Report a bug[8]

12.3.1. Form Authentication

This authentication scheme allows you to authenticate your users using a HTML Form element to capture user's credentials. If you're already familiar with the Java JEE FORM authentication method, you'll find this very similar.

To configure this authentication scheme for a specific path just do:

```
httpBuilder
    .forPath("/faces/*.xhtml")
        .authenticateWith()
            .form()
                .loginPage("/faces/login.xhtml")
                .errorPage("/faces/loginFailed.xhtml");
```

You will also need a login page with a HTML Form just like that:

```
<form method="POST" action="j_security_check">
    <input type="text" name="j_username"/>
    <input type="password" name="j_password"/>
    <input type="submit" name="login" value="Login"/>
</form>
```

By default, once the user is authenticated, PicketLink will always redirect the authenticated user to your application's context path. But sometimes you may want to restore the original request, the one used to start the authentication process. In this case, PicketLink will redirect the user to the original request once the authentication finishes. To enable this behavior just do:

```
httpBuilder
    .forPath("/faces/*.xhtml")
        .authenticateWith()
            .form()
                .loginPage("/faces/login.xhtml")
                .errorPage("/faces/loginFailed.xhtml")
                .restoreOriginalRequest();
```

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A41522-701489+%5BLatest%5D&comment=Title%3A+Authentication%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
```

Here, we used the **`restoreOriginalRequest()`** to enable this behavior.

Report a bug[9]

12.3.2. Basic Authentication

This authentication scheme allows you to authenticate your users using HTTP BASIC based on the IETF RFC standard.

To configure this authentication scheme for a specific path just do:

```
httpBuilder
    .forPath("/faces/*.xhtml")
        .authenticateWith()
            .basic()
                .realmName("My Application Realm");
```

If you not provide a **`realmName`**, PicketLink will use a default value, which is "PicketLink Default Realm".

Report a bug[10]

12.3.3. Digest Authentication

This authentication scheme allows you to authenticate your users using HTTP DIGEST based on the IETF RFC standard.

To configure this authentication scheme for a specific path just do:

```
httpBuilder
    .forPath("/faces/*.xhtml")
        .authenticateWith()
            .digest()
                .realmName("My Application Realm");
```

If you not provide a **`realmName`**, PicketLink will use a default value, which is "PicketLink Default Realm".

Report a bug[11]

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+38682-701493+%5BLatest%5D&comment=Title%3A+Form+Authentication%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=38682-701493+22+Aug+2014+04%3A29+en-US+%5BLatest%5D

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+38683-701498+%5BLatest%5D&comment=Title%3A+Basic+Authentication%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=38683-701498+22+Aug+2014+04%3A30+en-US+%5BLatest%5D

[11] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+38681-701520+%5BLatest%5D&comment=Title%3A+Digest+Authentication%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=38681-701520+22+Aug+2014+04%3A30+en-US+%5BLatest%5D

12.3.4. X.509 Authentication

This authentication scheme allows you to authenticate your users using HTTP X.509 based on the IETF RFC standard.

To configure this authentication scheme for a specific path just do:

```
httpBuilder
    .forPath("/faces/*.xhtml")
        .authenticateWith()
            .x509()
                .subjectRegex("CN=(.*?), ");
```

You can use the **`subjectRegex`** to provide a regular expression that will be used to extract the subject's name from the certificate. If not provided, PicketLink will try to extract the name from the subject DN considering only the CN value.

Report a bug[12]

12.3.5. Token Authentication

This authentication scheme allows you to authenticate your users based on a token. A token is usually a string representing a set of claims for an user. It allows you to enable your application as a **`Token Provider`**, responsible for issuing tokens.

Once the user is in possession of a token, it must be included on every single request in order to re-authenticate the user and create his security context before the request processing.

Note

Usually, when using tokens to authenticate your users, you would prefer to configure the **`Identity`** Bean as stateless. Take a look at *Section 2.2.1, "Stateful or Stateless Authentication"* for more details.

By default, tokens are issued based on a HTTP BASIC authentication. This is the primary authentication method used in order to check for user credentials before issuing a token. If the authentication is successful, PicketLink will ask a **`Token Provider`** to issue a new token and write it to the response using a JSON format.

To configure this authentication scheme for a path, just do:

```
httpBuilder
    .forPath("/authenticate")
        .authenticateWith()
            .token();
```

[12] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+38679-701531+%5BLatest%5D&comment=Title%3A+X.509+Authentication%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=38679-701531+22+Aug+2014+04%3A31+en-US+%5BLatest%5D

Before using this authentication scheme, you must provide a representation for your token and also a token provider, responsible for issuing tokens. Let's start by defining a simple token representation:

```
public class MyToken extends AbstractToken {

    private String token;

    public MyToken(String token) {
        super(token);
    }

    @Override
    public String getSubject() {
        return this.token;
    }
}
```

This is the most simple example about how to represent a token. It is just a string value where the token itself contains the user identifier. PicketLink allows you to support any token format, you can even use JSON Web Token and JOSE specifications to provide a better representation for your tokens. In real world use cases, you would prefer a format where the token has an unique identifier, expiration time and a set of claims representing the information for your users such as: username, identifier, roles and so forth.

Once you define your token format, you must provide a token provider.

```
@Stateless
public class MyTokenProvider implements Token.Provider<MyToken> {

    @Inject
    private PartitionManager partitionManager;

    @Override
    public MyToken issue(Account account) {
        MyToken token = new MyToken(account.getId());
        IdentityManager identityManager = getIdentityManager(account);

        identityManager.updateCredential(account, token);

        return token;
    }

    @Override
    public MyToken renew(Account account, MyToken renewToken) {
        IdentityManager identityManager = getIdentityManager(account);
        TokenCredentialStorage tokenStorage = getCurrentToken(account, identityManager);

        if (tokenStorage.getToken().equals(renewToken.getToken())) {
            invalidate(account);
            return issue(account);
        }

        return null;
    }

    @Override
    public void invalidate(Account account) {
        IdentityManager identityManager = getIdentityManager(account);

        identityManager.removeCredential(account, TokenCredentialStorage.class);
    }

    @Override
    public Class<MyToken> getTokenType() {
```

```
        return MyToken.class;
    }

    private TokenCredentialStorage getCurrentToken(Account account, IdentityManager
 identityManager) {
        return identityManager.retrieveCurrentCredential(account,
 TokenCredentialStorage.class);
    }

    private IdentityManager getIdentityManager(Account account) {
        return this.partitionManager.createIdentityManager(account.getPartition());
    }
}
```

This is a very simple example of a **`Token.Provider`**. Token providers are responsible for issuing, invalidating and renewing tokens. In this case, we're issuing a token where its value is the id of a specific **`Account`**. In real world use cases, you'll prefer a better format such as JWT.

Note

In this case we're marking the token provider as an EJB by using the **`@Stateless`** annotation. The reason for that is that if you are using the JPA Identity Store you must initiate or join a transaction before persisting data to the database. When using an EJB all transaction management is done by the container. Considering this is just an exmaple about how to use a specific functionality, we are trying to keep the code simple without managing transaction manually.

To the this configuration you can start by sending a request to the */authenticate* path as follows:

```
POST /picketlink-forge-app/authenticate HTTP/1.1
Host: localhost:8080
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Cache-Control: no-cache
```

The request above is trying to invoke the given path without provide any credentials. In this case, PicketLink will respond with a **`401`** HTTP Status Code, teeling that authentication is required. The response also contains a specific header to indicate which type of authentication is required.

```
Content-Length →1062
Content-Type →text/html;charset=utf-8
Date →Thu, 21 Aug 2014 12:58:51 GMT
Server →Apache-Coyote/1.1
WWW-Authenticate →Token
```

As you can see, the server is asking for a **`Token`** by setting the **`WWW-Authenticate`** header.

Now, let's get a token from the server. By default, PicketLink performs a HTTP Basic authentication in order to validate the identity of an user before issuing a token. That is what we call, the *primary authentication scheme*. You can always change this behavior and use whatever you want to validate the user identity prior to issue tokens.

```
POST /picketlink-forge-app/authenticate HTTP/1.1
Host: localhost:8080
```

```
Authorization: Basic cGlja2V0bGluazpwaWNrZXRsaW5r
Cache-Control: no-cache
```

The request above is sending a HTTP Basic credential to the server. If the server successfuly validates the given credential then it will finally respond with the token:

```
{"authctoken":"4761e6f7-847d-4a7a-afc1-1a899c3d5d61"}
```

By default, PicketLink will just write to the response body a JSON containing the token. Useful if you are in a HTML5 application using AJAX to authenticate your users. But you can also change this behavior if you want to.

This authentication scheme also provides a special behavior when handling AJAX requests. As you might know, some browsers display an authentication dialog when receiving a 401 status code even if you are using AJAX to send requests. PicketLink tries to avoid this issue by just checking the presence of the **`X-Requested-With`** header in the request. Most AJAX libraries such as JQuery and AngularJS do use this header when sending AJAX requests. In this case, PicketLink will not respond with a 401, but with a 403 status code.

Report a bug[13]

12.3.6. Write Your Own Authentication Scheme

If none of the built-in authentication schemes fullfil your requirements, you can always provide your own implementation.

To configure your custom authentication scheme you just need to:

```
httpBuilder
    .forPath("/authenticate")
        .authenticateWith()
            .scheme(MyCustomAuthenticationScheme.class);
```

Authentication scheme implementations are simple Java types that implements the **`org.picketlink.http.authentication.HttpAuthenticationScheme`** interface.

```
public class MyCustomAuthenticationScheme implements HttpAuthenticationScheme {

    @Override
    public void initialize(AuthenticationSchemeConfiguration config) {

    }

    @Override
    public void extractCredential(HttpServletRequest request, DefaultLoginCredentials creds)
 {
    // use this method to extract credentials from request and set them into
 DefaultLoginCredentials
    }
```

[13] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+38680-701551+%5BLatest%5D&comment=Title%3A+Token+Authentication%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=38680-701551+22+Aug+2014+04%3A32+en-US+%5BLatest%5D

```
    @Override
    public void challengeClient(HttpServletRequest request, HttpServletResponse response) {
    // use this method to challenge client for credentials
    }

    @Override
    public void onPostAuthentication(HttpServletRequest request, HttpServletResponse
 response) {
    // use this method to perform any logic after an authentication attempt
    }
}
```

Authentication scheme types are just regular CDI beans.

Report a bug[14]

12.4. Authorization

PicketLink supports different authorization methods for paths, they are:

- **RBAC**, Role-Based Access Control
- **GBAC**, Group-Based Access Control
- **`Realm-Based Access Control`**
- **`Expression Language`**
- **`Write Your Authorization Method`**

When you configure the authorization policies to a specific path you just need to provide any of the available options available from the **`authorizeWith()`** method provided by the **`HttpSecurityBuilder`**.

```
httpBuilder
    .forPath("/admin/*")
        .authorizeWith()
            .role("Administrator")
```

In the next sections we'll cover each of them in more details.

Report a bug[15]

12.4.1. Role-Based Authorization

This method allows you to perform authorization based on the roles granted for your users. Access is granted only if they were granted with a specific set of roles.

[14] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41512-701552+%5BLatest%5D&comment=Title%3A+Write+Your+Own+Authentication+Scheme%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41512-701552+22+Aug+2014+04%3A32+en-US+%5BLatest%5D

[15] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A41508-701598+%5BLatest%5D&comment=Title%3A+Authorization%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

To configure this method of authorization for a specific path just do:

```
httpBuilder
    .forPath("/admin/*")
        .authorizeWith()
            .role("Administrator");
```

Here, the **`role`** method expects one or more role names.

Report a bug[16]

12.4.2. Group-Based Authorization

This method allows you to perform authorization based on the groups an user belongs to. Access is only granted if he is member of a set of groups.

To configure this method of authorization for a specific path just do:

```
httpBuilder
    .forPath("/admin/*")
        .authorizeWith()
            .group("Administrators");
```

Here, the **`group`** method expects one or more group names. When checking if an user is member of a Group, PicketLink will consider the parent-child relationship of each group defined in the configuration.

Report a bug[17]

12.4.3. Realm-Based Authorization

This method allows you to perform authorization based on the realm an user belongs to. Access is only granted if the user belongs to a specific **`Partition`**.

To configure this method of authorization for a specific path just do:

```
httpBuilder
    .forPath("/acme/*")
        .authorizeWith()
            .realm("Acme");
```

Here, the **`realm`** method expects one or more realm names. In this case, only users from a **`Partition`** with name **`Acme`** are allowed to access a given path.

[16] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41507-701585+%5BLatest%5D&comment=Title%3A+Role-Based+Authorization%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41507-701585+22+Aug+2014+04%3A33+en-US+%5BLatest%5D

[17] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41516-701596+%5BLatest%5D&comment=Title%3A+Group-Based+Authorization%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41516-701596+22+Aug+2014+04%3A34+en-US+%5BLatest%5D

You can also specify the **`Partition`** type instead. For example:

```
httpBuilder
    .forPath("/acme/*")
        .authorizeWith()
            .realm(Acme.class.getName());
```

Where **`Acme`** is a **`Partition`** type.

Report a bug[18]

12.4.4. Expression-Based Authorization

This method allows you to perform authorization based on an expression using Java EL .

To configure this method of authorization for a specific path just do:

```
httpBuilder
    .forPath("/acme/*")
        .authorizeWith()
            .expression("#{identity.account.partition.name == 'Acme'}");
```

Here, the **`expression`** method expects one or more expressions. In this case, we're using the **`Identity`** Bean to retrieve the authenticated account and check if it belongs to the **`Acme`** partition. Pretty much the same rule we provided when using the **`Realm-Based Authorization`** method.

When writing expressions, you are allowed to use any of the available function provided by PicketLink. For a complete list, take a look at *Section 11.8, "Using EL-Based Expresions"*.

PicketLink provides some basic support for URL rewriting based on EL expressions. You can configure a specific path with a dynamic authorization check using EL expressions. Let's suppose you have the following configuration for a given path:

```
httpBuilder
    .forPath("/company/{identity.account.partition.name}/{identity.account.id}/*")
        .authorizeWith()
            .expression("#{identity.account.partition.name}", "#{identity.account.id}");
```

When you send a request to your application using */company/{identity.account.partition.name}/{identity.account.id}/**, PicketLink will automatically evaluate each expression to build the real path.

```
/company/default_partition/1/*
```

Once the path is rewritten, PicketLink will check if the authorization expressions matches the path. In this case, if the user tries to access a path using the identifier from another user, PicketLink will block and deny the request.

[18] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41518-701609+%5BLatest%5D&comment=Title%3A+Realm-Based+Authorization%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41518-701609+22+Aug+2014+04%3A35+en-US+%5BLatest%5D

Report a bug[19]

12.4.5. Write Your Own Path Authorizer

If none of the built-in authorization methods fullfil your requirements, you can always provide your own implementation.

To configure your custom authorization method for a specific path you just need to:

```
httpBuilder
    .forPath("/custom/protected/*")
        .authorizeWith()
            .authorizer(MyCustomPathAuthorizer.class);
```

In order to provide your own authorization logic you must provide an implementation for the **`org.picketlink.http.authorization.PathAuthorizer`** interface.

```
public static class CustomPathAuthorizer implements PathAuthorizer {

        @Override
        public boolean authorize(PathConfiguration pathConfiguration, HttpServletRequest
 request, HttpServletResponse response) {
            // perform authorization
        }
    }
```

`org.picketlink.http.authorization.PathAuthorizer` types are just regular CDI beans.

Report a bug[20]

12.5. Logout

PicketLink allows you to configure a path to logout users. To configure a logout path just do:

```
httpBuilder
    .forPath("/logout")
        .logout()
```

Here, the **`logout`** method will mark the given path as a point of logout. In this case, when a request arrives to this path PicketLink will logout the user.

In some cases you may want to redirect the user to a specific path after a successful logout. For that you just need to:

```
httpBuilder
```

[19] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41513-701624+%5BLatest%5D&comment=Title%3A+Expression-Based+Authorization%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41513-701624+22+Aug+2014+04%3A36+en-US+%5BLatest%5D

[20] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41529-701627+%5BLatest%5D&comment=Title%3A+Write+Your+Own+Path+Authorizer%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41529-701627+22+Aug+2014+04%3A36+en-US+%5BLatest%5D

```
    .forPath("/logout")
        .logout()
        .redirectTo("/goodbye.html");
```

Report a bug[21]

12.6. Servlet API Integration

PicketLink provides a seamless integration with the Java Servlet API. As you might know, the Servlet API provides some useful methods to authenticate, logout and check for user's roles. This is usually performed by invoking specific methods on the **`HttpServletRequest`**.

For example, let's say you want to authenticate an user using his username and password. The Servlet API provides a specific method that you can use as follows:

```
HttpServletRequest request = // get request

request.login("john", "password");
```

Once the user is authenticated, you are able to get the principal from the request as follows:

```
Principal principal = request.getUserPrincipal();
```

The same applies to logout. You can easily logout an user by just:

```
request.logout();
```

The Servlet API also provides a method to check users roles.

```
request.isUserInRole("Administrator);
```

Report a bug[22]

[21] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41510-701633+%5BLatest%5D&comment=Title%3A+Logout%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41510-701633+22+Aug+2014+04%3A37+en-US+%5BLatest%5D

[22] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41606-705497+%5BLatest%5D&comment=Title%3A+Servlet+API+Integration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41606-705497+02+Sep+2014+01%3A26+en-US+%5BLatest%5D

PicketLink Subsystem

13.1. Overview

The PicketLink Subsystem extends JBoss Application Server to introduce some new capabilities, providing a infrastructure to deploy and manage PicketLink deployments and services.

In a nutshell, the most important capabilities are:

- A rich domain model supporting the configuration of PicketLink Federation (specially SAML-based applications) deployments and Identity Management services.
- Minimal configuration for deployments. Part of the configuration is done automatically with some hooks for customizations.
- Minimal dependencies for deployments. All PicketLink dependencies are automatically set from modules.
- Configuration management using JBoss Application Server Management API. It can be managed in different ways: HTTP/JSON, CLI, Native DMR, etc.
- Identity Management Services are exposed in JNDI and are fully integrated with CDI.
- Applications don't need to change when moving between different environments such as development, testing, staging or production. All the configuration is defined outside the application.
- Users should learn a single and consolidated schema.

Important

The subsystem is only available in JBoss Enterprise Application Platform 6.3+ and WildFly 9.

Report a bug[1]

13.2. Installation and Configuration

You must change your **`standalone.xml`** or **`domain.xml`**, inside your JBoss AS installation, with the following extension and subsystem:

```
<extensions>
  ...

  <!-- Add the PicketLink extension -->
  <extension module="org.wildfly.extension.picketlink"/>

</extensions>
```

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29749-673957+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29749-673957+18+Jun+2014+00%3A37+en-US+%5BLatest%5D

```
<profile>

  <!-- Add the PicketLink Identity Management Subsystem -->
  <subsystem xmlns="urn:jboss:domain:picketlink-identity-management:1.0"/>

  <!-- Add the PicketLink Federation Subsystem -->
  <subsystem xmlns="urn:jboss:domain:picketlink-federation:1.0"/>

</profile>
```

The next sections will describe how to configure the subsystems individually.

Report a bug[2]

13.3. Configuring the PicketLink Dependencies for your Deployment

Please, take a look at *Section 1.8, "Referencing PicketLink from JBoss Modules"* for more details.

Report a bug[3]

13.4. Domain Model

Each subsystem provides a domain model that allows you to configure their respective services using the **`standalone.xml`** or **`domain.xml`** inside your JBoss AS installation. The domain model is very easy to understand if you are already familiar with the PicketLink configuration.

```
<!-- An example of the PicketLink Federation configuration -->
<subsystem xmlns="urn:jboss:domain:picketlink-federation:1.0">
   <federation alias="federation-with-signatures">

     <saml token-timeout="4000" clock-skew="0"/>

     <key-store url="/jbid_test_keystore.jks" passwd="changeit" sign-key-alias="localhost"
 sign-key-passwd="changeit"/>

     <identity-provider url="http://localhost:8080/idp-sig/" alias="idp-sig.war" security-
domain="idp" supportsSignatures="true">

       <trust>
         <trust-domain name="localhost" cert-alias="localhost"/>
         <trust-domain name="127.0.0.1" cert-alias="localhost"/>
       </trust>

     </identity-provider>

     <service-providers>
```

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29750-673958+%5BLatest%5D&comment=Title%3A+Installation+and+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29750-673958+18+Jun+2014+00%3A49+en-US+%5BLatest%5D

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29751-676193+%5BLatest%5D&comment=Title%3A+Configuring+the+PicketLink+Dependencies+for+your+Deployment%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29751-676193+24+Jun+2014+22%3A43+en-US+%5BLatest%5D

```
      <service-provider alias="sales-post-sig.war" security-domain="sp" url="http://
localhost:8080/sales-post-sig/"/>

      <service-provider alias="sales-redirect-sig.war" security-domain="sp" url="http://
localhost:8080/sales-redirect-sig/" supportsSignatures="true" />

    </service-providers>
  </federation>

</subsystem>

<!-- A configuration using a JPA-based identity store. The store is configured using a
 existing datasource. -->
<subsystem xmlns="urn:jboss:domain:picketlink-identity-management:1.0">

 <partition-manager jndi-name="picketlink/JPADSBasedPartitionManager"
 name="jpa.ds.based.partition.manager">

  <identity-configuration name="jpa.config">

   <jpa-store data-source="jboss/datasources/ExampleDS">
    <supported-types supports-all="true"/>
   </jpa-store>

  </identity-configuration>

 </partition-manager>

</subsystem>
```

A complete reference for each XML Schema defined by these subsystems can be found at:

- **`PicketLink Federation`** - *https://github.com/wildfly/wildfly/blob/master/build/src/main/resources/docs/schema/wildfly-picketlink-federation_1_0.xsd*
- **`PicketLink Identity Management`** - *https://github.com/wildfly/wildfly/blob/master/build/src/main/resources/docs/schema/wildfly-picketlink-idm_1_0.xsd*

Report a bug[4]

13.5. Identity Management

This subsystem provides a domain model that allows you to configure PicketLink Identity Management Services using the **`standalone.xml`** or **`domain.xml`**. Basically, what the subsystem does is parse the configuration, automatically build a **`PartitionManager`** and expose it via JNDI for further access.

With the subsystem you can :

- Externalize and centralize the IDM configuration for deployments.
- Define multiple configuration for identity management services.
- Expose the **`PartitionManager`** via JNDI for further access.

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29752-673964+%5BLatest%5D&comment=Title%3A+Domain+Model%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29752-673964+18+Jun+2014+01%3A15+en-US+%5BLatest%5D

- If using CDI, inject the **`PartitionManager`** instances using the **`Resource`** annotation.
- If using CDI, use the PicketLink IDM alone without requiring the base module dependencies.

The IDM domain model is an abstraction for all PicketLink IDM configuration, providing a single schema from which all configuration can be defined. If you're already familiar with the Configuration API, you'll find the syntax pretty simple and intuitive.

```
<subsystem xmlns="urn:jboss:domain:picketlink-identity-management:1.0">

 <!-- A complete configuration using a file-based identity store. -->
        <partition-manager jndi-name="picketlink/FileCompletePartitionManager"
 name="file.complete.partition.manager">
            <identity-configuration name="file.config">
                <file-store relative-to="jboss.server.data.dir" working-dir="pl-idm-complete"
 always-create-files="true" async-write="true"
                            async-write-thread-pool="10">
                    <supported-types supports-all="true"/>
                </file-store>
            </identity-configuration>
        </partition-manager>

 <!-- A configuration using a JPA-based identity store. The store is configured using a
 existing datasource. -->
        <partition-manager jndi-name="picketlink/JPADSBasedPartitionManager"
 name="jpa.ds.based.partition.manager">
            <identity-configuration name="jpa.config">
                <jpa-store data-source="jboss/datasources/ExampleDS">
                    <supported-types supports-all="true"/>
                </jpa-store>
            </identity-configuration>
        </partition-manager>

</subsystem>
```

Note

If you are looking for more examples about how to use the domain model, take a look at **`JBOSS_HOME`**/docs/examples/configs/standalone-picketlink.xml.

Most of the configuration are known if you are familiar with the PicketLink IDM configuration. But the domain model provides some additional configuration in order to allow deployments to access the configured identity management services. Basically, each configuration must have a:

- **jndi-url**, that defines where the **`PartitionManager`** should be published in the JNDI tree for further access.
- **name**, the name of the configuration to allow other subsystems to inject the Identity Management Services using the MSC injection infrastructure.

The rest of the configuration is very similar with how you use the Configuration API to programmaticaly build the IDM configuration. For a complete description of the domain model elements, please take a look at the XML Schema.

Report a bug[5]

13.5.1. <code xmlns="http://docbook.org/ns/docbook">JPAIdentityStore</code>

In order to provide a better and easy integration with the container, the **`JPAIdentityStore`** configuration provides some additional configuration to let you configure how the **`EntityManagerFactory`** is built or used by the **`JPAIdentityStore`**.

Report a bug[6]

13.5.1.1. Using a DataSource JNDI Url

When you specify a DataSource JNDI url, the subsystem will automatically build a **`EntityManagerFactory`** using a default configuration. This is the fastest way to get a JPA Identity Store up and running, specially if you just want to use the *Section 5.1, "Basic Identity Model"* provided by PicketLink.

The DataSource JNDI url can be specified using the **data-source** attribute as follows:

```
<subsystem xmlns="urn:jboss:domain:picketlink-identity-management:1.0">

<!-- A configuration using a JPA-based identity store. The store is configured using a
 existing datasource. -->
        <partition-manager jndi-name="picketlink/JPADSBasedPartitionManager"
 name="jpa.ds.based.partition.manager">
            <identity-configuration name="jpa.config">
                <jpa-store data-source="jboss/datasources/ExampleDS">
                    <supported-types supports-all="true"/>
                </jpa-store>
            </identity-configuration>
        </partition-manager>

</subsystem>
```

This configuration option is very handy if you want to use the Basic Model provided by PicketLink.

Report a bug[7]

13.5.1.2. Using a `EntityManagerFactory` JNDI Url

Sometimes you may need more control over the JPA Persistence Unit configuration. In this case you can use the **entity-manager-factory** attribute to specify where your previously built **`EntityManagerFactory`** is located.

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29753-673987+%5BLatest%5D&comment=Title%3A+Identity+Management%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29754-620360+%5BLatest%5D&comment=Title%3A+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EJPAIdentityStore%3C%2Fcode%3E%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[7] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29755-673989+%5BLatest%5D&comment=Title%3A+Using+a+DataSource+JNDI+Url%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29755-673989+18+Jun+2014+02%3A57+en-US+%5BLatest%5D

```
<subsystem xmlns="urn:jboss:domain:picketlink-identity-management:1.0">

 <!-- A configuration using a JPA-based identity store. The store is configured using a
 existing JPA EntityManagerFactory, obtained via JNDI. -->
        <partition-manager jndi-name="picketlink/JPAEMFBasedPartitionManager"
 name="jpa.emf.based.partition.manager">
            <identity-configuration name="jpa.config">
                <jpa-store entity-manager-factory="jboss/MyEntityManagerFactory">
                    <supported-types>
                        <supported-type code="Partition"/>
                        <supported-type code="IdentityType"/>
                        <supported-type code="Relationship"/>
                    </supported-types>
                </jpa-store>
          </identity-configuration>
        </partition-manager>

</subsystem>
```

Report a bug[8]

13.5.1.3. Using a JBoss Module

The JPA Identity Store configuration allows you to specify a JBoss Module from where the JPA Persistence Unit and mapped entities will be loaded from.

This configuration can be done using two attributes:

- **entity-module**, the module name where the JPA Persistence Unit and all mapped entities are located.
- **entity-module-unit-name**, the name of the JPA Persistence Unit name. If you don't provide a name the subsystem will use **identity**.

```
<subsystem xmlns="urn:jboss:domain:picketlink-identity-management:1.0">

 <!-- A configuration using a JPA-based identity store. The store is configured using a
 existing JPA Persistence Unit from a static module. -->
        <partition-manager jndi-name="picketlink/JPAEMFModulePartitionManager"
 name="jpa.emf.modules.partition.manager">
            <identity-configuration name="jpa.config">
                <jpa-store entity-module="my.module" entity-module-unit-name="my-persistence-
unit-name">
                    <supported-types supports-all="true"/>
                </jpa-store>
          </identity-configuration>
        </partition-manager>

  </subsystem>
```

Report a bug[9]

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29756-673990+%5BLatest%5D&comment=Title%3A+Using+a+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EEntityManagerFactory%3C%2Fcode%3E+JNDI+Url%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29756-673990+18+Jun+2014+02%3A58+en-US+%5BLatest%5D

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29757-673969+%5BLatest

13.5.2. Usage Examples

If you want to have your deployment properly configured with the PicketLink Identity Management Services, you should add a **`META-INF/jboss-deployment-structure.xml`** file to your deployment as follows:

```
<jboss-deployment-structure>
 <deployment>
  <dependencies>
   <module name="org.picketlink.core.api" meta-inf="import"/>
   <module name="org.picketlink.core" meta-inf="import"/>
   <module name="org.picketlink.idm.api" />

   <!-- if you're using a JPA Identity Store and the Basic Model, you must provide this
 dependency. -->
   <module name="org.picketlink.idm.schema" />
  </dependencies>
 </deployment>
</jboss-deployment-structure>
```

Note

When you're configuring the PicketLink dependencies using the **`META-INF/jboss-deployment-structure.xml`** file you don't need to ship the libraries inside your deployment. All the necessary dependencies are automatically resolved and configured.

Report a bug[10]

13.5.2.1. Injecting a `PartitionManager` using `Resource` annotation

```
@Resource(mappedName="picketlink/JPADSBasedPartitionManager")
private PartitionManager jpaDSBasedPartitionManager;
```

Report a bug[11]

13.5.2.2. Producing a PartitionManager

You may want to use a **`PartitionManager`** instance built by the subsystem. For that, you just need to:

%5D&comment=Title%3A+Using+a+JBoss+Module%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29757-673969+18+Jun+2014+01%3A30+en-US+%5BLatest%5D

[10] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29758-673972+%5BLatest%5D&comment=Title%3A+Usage+Examples%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[11] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29759-673971+%5BLatest%5D&comment=Title%3A+Injecting+a+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EPartitionManager%3C%2Fcode%3E+using+%3Ccode+xmlns%3D%22http%3A%2F%2Fdocbook.org%2Fns%2Fdocbook%22%3EResource%3C%2Fcode%3E+annotation%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29759-673971+18+Jun+2014+01%3A32+en-US+%5BLatest%5D

```
public static class SecurityConfiguration {

 @Resource(mappedName = "picketlink/JPADSBasedPartitionManager")
 @Produces
 private PartitionManager jpaDSBasedPartitionManager;

}
```

Report a bug[12]

13.6. Federation

All the configuration is external from applications where there is no need to add or change configuration files inside the application being deployed. The subsystem is responsible for configure the applications being deployed accordingly with the configurations defined in the domain model:

- The configuration in **`picketlink.xml`** is automatically created. No need to have this file inside your deployment.
- The PicketLink Authenticators for Identity Providers and Service Providers are automatically registered. No need to have a **`jboss-web.xml`** file inside your deployment.
- The PicketLink dependencies are automatically configured. No need to have a **`META-INF/jboss-deployment-structure.xml`** inside your deployment defining the **`org.picketlink`** module as a dependency.
- The Security Domain is automatically configured using the configurations defined in the domain model. No need to have a **`WEB-INF/jboss-web.xml`** file inside your deployment.

The table bellow summarizes the main differences between the traditional configuration and the subsystem configuration for PicketLink applications:

Configuration	Old Configuration	Subsystem Configuration
WEB-INF/picketlink.xml	Required	Not required. If present, the configuration from the domain model is going to be used instead.
WEB-INF/jboss-web.xml	Required	Not required. The PicketLink Authenticators and the Security Domain is read from the domain model.
META-INF/jboss-deployment-structure.xml	Required	Not required. When the PicketLink Extension/ Subsystem is enabled, the dependency to the org.picketlink module is automatically configured.

[12] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29760-673973+%5BLatest%5D&comment=Title%3A+Producing+a+PartitionManager%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29760-673973+18+Jun+2014+01%3A40+en-US+%5BLatest%5D

Report a bug[13]

13.6.1. The Federation concept (Circle of Trust)

When using the PicketLink subsystem to configure and deploy your identity providers and service providers, all of them are grouped in a Federation.

A Federation can be understood as a Circle of Trust (CoT) from which applications share common configurations (certificates, saml specific configurations, etc) and where each participating domain is trusted to accurately document the processes used to identify a user, the type of authentication system used, and any policies associated with the resulting authentication credentials.

Each federation has one Identity Provider and many Service Providers. You do not need to specify for each SP the IDP that it trusts, because this is defined by the federation.

Report a bug[14]

13.6.2. Federation Domain Model

The domain model is an abstraction for all PicketLink Federation configuration, providing a single schema from which all configurations can be defined for Identity Providers or Service Providers, for example.

The example bellow shows how the domain model can used to configure an Identity Provider and a Service Provider.

```
<subsystem xmlns="urn:jboss:domain:picketlink:1.0">
  <federation name="federation-without-signatures">

    <saml token-timeout="4000" clock-skew="0" />

    <identity-provider name="idp.war" security-domain="idp" support-signatures="false"
 url="http://localhost:8080/idp/">
        <trust>
            <trust-domain name="localhost" />
            <trust-domain name="mycompany.com2" />
            <trust-domain name="mycompany.com3" />
            <trust-domain name="mycompany.com4" />
        </trust>
        <handlers>
            <handler class-name="com.mycompany.CustomHandler">
                <handler-parameter name="param1" value="paramValue1"/>
                <handler-parameter name="param2" value="paramValue2"/>
                <handler-parameter name="param3" value="paramValue3"/>
            </handler>
        </handlers>
    </identity-provider>

    <service-providers>
```

[13] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29761-689822+%5BLatest%5D&comment=Title%3A+Federation%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[14] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29762-620368+%5BLatest%5D&comment=Title%3A+The+Federation+concept+%28Circle+of+Trust%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29762-620368+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

```
        <service-provider name="sales.war" post-binding="true" security-domain="sp"
 url="http://localhost:8080/sales/" support-signatures="false">
            <handlers>
                <handler class-name="com.mycompany.CustomHandler">
                    <handler-parameter name="param1" value="paramValue1"/>
                    <handler-parameter name="param2" value="paramValue2"/>
                    <handler-parameter name="param3" value="paramValue3"/>
                </handler>
            </handlers>
        </service-provider>
        <service-provider name="employee.war" post-binding="true" security-domain="sp"
 url="http://localhost:8080/employee/" support-signatures="false" />
    </service-providers>
  </federation>
</subsystem>
```

Note

If you are looking for more examples about how to use the domain model, take a look at **JBOSS_HOME**/docs/examples/configs/standalone-picketlink.xml.

Report a bug[15]

13.6.3. Usage Examples

This section will guide you through the basic steps to get an Identity Provider and a Service Provider working using the subsystem configuration.

Before starting, make sure you have the *Section 1.7, "PicketLink Installer"* properly configured.

Build the quickstarts and copy the file and copy the picketlink-quickstarts/picketlink-federation-saml-idp-basic/target/picketlink-federation-saml-idp-basic.war and picketlink-quickstarts/picketlink-federation-saml-sp-post-basic/target/picketlink-federation-saml-sp-post-basic.war to ${JBOSS.HOME.dir}/standalone/deployments.

Open both files and remove the following configuration files:

- WEB-INF/picketlink.xml
- META-INF/jboss-deployment-structure.xml
- WEB-INF/jboss-web.xml

[15] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29763-689820+%5BLatest%5D&comment=Title%3A+Federation+Domain+Model%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29763-689820+01+Aug+2014+23%3A34+en-US+%5BLatest%5D

Important

Don't forget to configure the security domains for both applications.

Open the standalone.xml and add the following configuration for the PicketLink subsystem:

```
<subsystem xmlns="urn:jboss:domain:picketlink-federation:1.0">
  <federation name="example-federation">
      <!-- Identity Provider configuration -->
      <identity-provider name="picketlink-federation-saml-idp-basic.war" security-
domain="idp" url="http://localhost:8080/idp/">
          <trust>
              <trust-domain name="localhost" />
          </trust>
      </identity-provider>

      <!-- Service Provider configuration -->
      <service-providers>
          <service-provider name="picketlink-federation-saml-sp-post-basic.war" security-
domain="sp" url="http://localhost:8080/sales-post/" />
      </service-providers>
  </federation>
</subsystem>
```

To make sure that everything is ok, please start JBoss AS and try to access the sales application. You should be redirected to the IdP application.

Report a bug[16]

13.6.4. Metrics and Statistics

Metrics and statistics can be collected from applications deployed using the PicketLink subsystem. This means you can get some useful information about how your Identity Providers and Service providers are working.

- How many SAML assertions were issued by your identity provider ?
- How many times your identity provider respond to service providers ?
- How many SAML assertions were expired ?
- How many authentications are done by your identity provider ?
- How many errors happened ? Trusted Domain errors, signing errors, etc.

To query those metrics and statistics you can use JBoss CLI as follows:

```
[standalone@localhost:9999 federation=example-federation] ./identity-provider=idp.war:read-
resource(include-runtime=true)
```

[16] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29764-689821+%5BLatest%5D&comment=Title%3A+Usage+Examples%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29764-689821+01+Aug+2014+23%3A36+en-US+%5BLatest%5D

```
{
    "outcome" => "success",
    "result" => {
         "alias" => "idp.war",
         "created-assertions-count" => "1",
         "error-response-to-sp-count" => "0",
         "error-sign-validation-count" => "0",
         "error-trusted-domain-count" => "0",
         "expired-assertions-count" => "0",
         "external" => false,
         "handler" => undefined,
         "login-complete-count" => "0",
         "login-init-count" => "0",
         "response-to-sp-count" => "3",
         "security-domain" => "idp",
         "strict-post-binding" => false,
         "supportsSignatures" => false,
         "url" => "http://localhost:8080/idp",
         "trust-domain" =>{"localhost" => undefined}
    }
}
```

Report a bug[17]

[17] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29765-673982+%5BLatest%5D&comment=Title%3A+Metrics+and+Statistics%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29765-673982+18+Jun+2014+02%3A45+en-US+%5BLatest%5D

Federation

14.1. Overview

In this chapter, we look at PicketLink **`single sign on (SSO)`** and **`trust`** features. We describe **`SAML`** SSO in detail.

Report a bug[1]

14.2. SAML SSO

SAML is an **`OASIS Standards Consortium`** standard for single sign on. PicketLink suppors **`SAML v2.0`** and **`SAML v1.1`**.

PicketLink contains support for the following profiles of SAML specification.

- SAML Web Browser SSO Profile.
- SAML Global Logout Profile.

Report a bug[2]

14.3. SAML Web Browser Profile

PicketLink supports the following standard bindings:

- SAML HTTP Redirect Binding
- SAML HTTP POST Binding

Report a bug[3]

14.4. PicketLink SAML Specification Support

PicketLink aims to provide support for both SAML v1.1 and v2.0 specifications. The emphasis is on SAMLv2.0 as v1.1 is deprecated.

Report a bug[4]

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29767-620373+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29767-620373+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29768-620374+%5BLatest%5D&comment=Title%3A+SAML+SSO%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29768-620374+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29769-620375+%5BLatest%5D&comment=Title%3A+SAML+Web+Browser+Profile%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29769-620375+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29770-620376+%5BLatest%5D&comment=Title%3A+PicketLink+SAML+Specification+Support%0A%0ADescribe+the+issue%3A%0A%0A

14.5. SAML v2.0

14.5.1. Which Profiles are supported ?

- SAML2 Web Browser Profile
- SAML2 Metadata Profile
- SAML2 Logout Profile

Report a bug[5]

14.5.2. Which Bindings are supported ?

The SAML v2 specification defines the concept of SAML protocol bindings (or just bindings). These bindings defines how SAML request-response messages are exchanged onto standard messaging or communication protocols. Currently, PicketLink support the following bindings:

- SAML HTTP Redirect Binding
- SAML HTTP POST Binding

Report a bug[6]

14.5.3. PicketLink Identity Provider (PIDP)

14.5.3.1. Introduction

The Identity Provider is the authoritative entity responsible for authenticating an end user and asserting an identity for that user in a trusted fashion to trusted partners.

Tip

Please look at the *PicketLink Quickstarts*[7] for the PicketLink Identity Provider web application. The quickstarts are useful resources where you can get configuration files.

%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29770-620376+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29771-620377+%5BLatest%5D&comment=Title%3A+Which+Profiles+are+supported+%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29771-620377+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[6] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29772-620378+%5BLatest%5D&comment=Title%3A+Which+Bindings+are+supported+%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29772-620378+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[7] https://docs.jboss.org/author/pages/viewpage.action?pageId=23986289

Report a bug[8]

14.5.3.2. How to create your own PicketLink Identity Provider

The best way to create your own Identity Provider implementation is using one of the examples provided by the PicketLink Quickstarts.

You should also take a look at the following documentations:

- *Section 14.5.3.4, "Identity Provider Configuration"*
- *Section 14.5.3.3, "Identity Provider Authenticators"*
- *Section 14.5.3.5, "Identity Stores"*

Report a bug[9]

14.5.3.3. Identity Provider Authenticators

14.5.3.3.1. Introduction

Error

Topic 29775 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 29775* for more detailed information.

14.5.3.3.2. Configuring an Authenthicator for a Identity Provider

The PicketLink Authenticator is basically a *Tomcat Valve*[10] (org.apache.catalina.authenticator.FormAuthenticator). The only thing you need to do is change the valves configuration for your application.

This configuration changes for each supported binding.

Report a bug[11]

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29773-626789+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29773-626789+01+Apr+2014+21%3A03+en-US+%5BLatest%5D

[9] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29774-620675+%5BLatest%5D&comment=Title%3A+How+to+create+your+own+PicketLink+Identity+Provider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29774-620675+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[10] http://tomcat.apache.org/tomcat-6.0-doc/config/valve.html

[11] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29776-626794+%5BLatest%5D&comment=Title%3A+Configuring+an+Authenthicator+for+a+Identity+Provider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.5.3.3.2.1. JBoss Application Server v7

In JBoss Application Server v7 the valves configuration are located inside the **WEB-INF/jboss-web.xml** file. Bellow is a example of how this file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
 <security-domain>idp</security-domain>
 <context-root>idp</context-root>
 <valve>
     <class-
name>org.picketlink.identity.federation.bindings.tomcat.idp.IDPWebBrowserSSOValve</class-
name>
 </valve>
</jboss-web>
```

The valve configuration is done using the **<valve>** element.

Report a bug[12]

14.5.3.3.2.2. JBoss Application Server v5 or v6

In JBoss Application Server v5 or v6, the valves configuration are located inside the **WEB-INF/context.xml** file. Bellow is a example of how this file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
<Valve
 className="org.picketlink.identity.federation.bindings.tomcat.idp.IDPWebBrowserSSOValve" />
</Context>
```

The valve configuration is done using the **<Valve>** element.

Report a bug[13]

14.5.3.3.2.3. Apache Tomcat 6

In Apache Tomcat 6 the valves configuration are located inside the **META-INF/context.xml** file. Bellow is a example of how this file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
<Valve
 className="org.picketlink.identity.federation.bindings.tomcat.idp.IDPWebBrowserSSOValve" />
</Context>
```

The valve configuration is done using the **<Valve>** element.

[12] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29777-620383+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v7%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29777-620383+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[13] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29778-620384+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v5+or+v6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29778-620384+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

Report a bug[14]

14.5.3.3.3. Built-in Authenticators

PicketLink provides default implementations for Service Provider Authenticators. The list bellow shows all the available implementations:

Name	Description
Section 14.5.3.3.4, "IDPWebBrowserSSOValve"	Default implementation for an Identity Provider Authenticator.

Report a bug[15]

14.5.3.3.4. IDPWebBrowserSSOValve

IDPWebBrowserSSOValve from PicketLink provides the core IDP functionality on JBoss Application Server or Apache Tomcat.

Report a bug[16]

14.5.3.3.4.1. Configuration

14.5.3.3.4.1.1. JBoss Application Server v6 and v5.x

Configure in WEB-INF/context.xml

Report a bug[17]

14.5.3.3.4.1.2. Apache Tomcat 5.5 and 6

Configure in META-INF/context.xml

Report a bug[18]

[14] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29779-620385+%5BLatest%5D&comment=Title%3A+Apache+Tomcat+6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29779-620385+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[15] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29780-620676+%5BLatest%5D&comment=Title%3A+Built-in+Authenticators%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29780-620676+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[16] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29781-620387+%5BLatest%5D&comment=Title%3A+IDPWebBrowserSSOValve%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[17] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29782-620388+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v6+and+v5.x%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29782-620388+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

[18] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29784-620390+%5BLatest%5D&comment=Title%3A+Apache+Tomcat+5.5+and+6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29784-620390+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

14.5.3.3.4.1.3. Example:

Example 14.1. context.xml

```
<Context>
<Valve
 className="org.picketlink.identity.federation.bindings.tomcat.idp.IDPWebBrowserSSOValve"
           signOutgoingMessages="false"
           ignoreIncomingSignatures="true"/>
</Context>
```

Report a bug[19]

14.5.3.3.4.2. Attributes

#	Name	Type	Objective	Since version
1	attributeList	String	a comma separated list of attribute keys IDP interested in	2.0
2	configProvider	String	an *optional* implementation of the SAMLConfigurationProvider interface. Provide the fully qualified name.	2.0
3	ignoreIncomingSignatures	boolean	if the IDP should ignore the signatures on the incoming messages Default: false	2.0 **Deprecated since 2.1.2.**
4	ignoreAttributeStatement	boolean	if the IDP should not generate attribute statements in response to Service Providers	2.0
5	signOutgoingMessages	boolean	Should the IDP sign the outgoing messages? Default: true	2.0 **Deprecated since 2.1.2.**
6	roleGenerator	String	optional fqn of a role generator Default: org.picketlink.identity. federation.bindings. tomcat.TomcatRoleGenerator	2.0 **Deprecated since 2.1.2.**
7	samlHandlerChainClass	String	fqn of a custom SAMLHandlerChain implementation	2.0 **Deprecated since 2.1.2.**
8	identityParticipantStack	String	fqn of a custom IdentityParticipantStack	2.0 **Deprecated since 2.1.2.**

[19] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29786-620392+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29786-620392+12+Mar+2014+12%3A31+en-US+%5BLatest%5D

Report a bug[20]

14.5.3.4. Identity Provider Configuration

14.5.3.4.1. Configuring a Identity Provider

To configure an application as a PicketLink Identity Provider you need to follow this steps:

1. Configure the web.xml.
2. Configure an *Section 14.5.4.4, "Service Provider Authenticators"* .
3. Configure a *Section 14.5.4.5, "Service Provider Security Domain"* for your application.
4. Configure ***PicketLink JBoss Module*** [21] as a dependency.
5. Create and configure a file named **WEB-INF/picketlink.xml** .

Report a bug[22]

14.5.3.4.2. Configuring the web.xml

Before configuring your application as an Identity Provider you need to add some configurations to your web.xml.

Let's start by defining a **security-constraint** element to restrict access to resources from unauthenticated users:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Manager command</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>manager</role-name>
    </auth-constraint>
</security-constraint>


<security-role>
    <description>
      The role that is required to log in to IDP Application
    </description>
    <role-name>manager</role-name>
</security-role>
```

[20] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29788-620394+%5BLatest%5D&comment=Title%3A+Attributes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29788-620394+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[21] https://docs.jboss.org/author/display/PLINK/JBoss+Modules

[22] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29789-626793+%5BLatest%5D&comment=Title%3A+Configuring+a+Identity+Provider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29789-626793+01+Apr+2014+21%3A04+en-US+%5BLatest%5D

As you can see above, we define that only users with a role named **manager** are allowed to access the protected resources. Make sure to give your users the same role you defined here, otherwise they will get a 403 HTTP status code.

The next step is define your *FORM* login configuration using the **login-config** element:

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>PicketLink IDP Application</realm-name>
    <form-login-config>
        <form-login-page>/jsp/login.jsp</form-login-page>
        <form-error-page>/jsp/login-error.jsp</form-error-page>
    </form-login-config>
</login-config>
```

Make sure you have inside your application the pages defined in the elements **form-login-page** and **form-error-page** .

Important

Please, make sure you have a welcome file page in your application. You can define it in your web.xml or simply create an **index.jsp** at the root directory of your application.

Report a bug[23]

14.5.3.4.3. The picketlink.xml configuration file

All the configuration for an especific Identity Provider goes at the WEB-INF/picketlink.xml file. This file is responsible to define the behaviour of the Authenticator. During the identity provider startup, the authenticator parses this file and configures itself.

Bellow is how the picketlink.xml file should looks like:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">


    <PicketLinkIDP xmlns="urn:picketlink:identity-federation:config:2.1">

        <IdentityURL>http://localhost:8080/idp/ </IdentityURL>


        <Trust>
            <Domains>locahost,mycompany.com</Domains>
        </Trust>

        <KeyProvider
 ClassName="org.picketlink.identity.federation.core.impl.KeyStoreKeyManager">

            <Auth Key="KeyStoreURL" Value="/jbid_test_keystore.jks" />
            <Auth Key="KeyStorePass" Value="store123" />
```

[23] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29790-620396+%5BLatest%5D&comment=Title%3A+Configuring+the+web.xml%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29790-620396+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

```
            <Auth Key="SigningKeyPass" Value="test123" />
            <Auth Key="SigningKeyAlias" Value="servercert" />

            <ValidatingAlias Key="localhost" Value="servercert" />
            <ValidatingAlias Key="127.0.0.1" Value="servercert" />

        </KeyProvider>

    </PicketLinkIDP>

    <PicketLinkSTS xmlns="urn:picketlink:identity-federation:config:1.0" TokenTimeout="1000"
 ClockSkew="1000">
        <TokenProviders>
            <TokenProvider
 ProviderClass="org.picketlink.identity.federation.core.saml.v2.providers.SAML20AssertionTokenProvider"
                TokenType="urn:oasis:names:tc:SAML:2.0:assertion" TokenElement="Assertion"
                TokenElementNS="urn:oasis:names:tc:SAML:2.0:assertion" />
        </TokenProviders>
    </PicketLinkSTS>

    <Handlers xmlns="urn:picketlink:identity-federation:handler:config:2.1">

        <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler" />
        <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler" />
        <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler" />
        <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler" />

     </Handlers>

</PicketLink>
```

Important

The schema for the picketlink.xml file is available here: *https://github.com/picketlink/picketlink/blob/master/modules/federation/src/main/resources/schema/config/picketlink_v2.1.xsd* .

Report a bug[24]

14.5.3.4.3.1. PicketLinkIDP Element

This element defines the basic configuration for the identity provider. The table bellow provides more information about the attributes supported by this element:

Name	Description	Value
AssertionValidity	Defines the timeout for the SAML assertion validity, in miliseconds.	Defaults to **300000** . *Deprecated. Use the*

[24] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29791-682436+%5BLatest%5D&comment=Title%3A+The+picketlink.xml+configuration+file%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

Name	Description	Value
		PicketLinkSTS element, instead.
RoleGenerator	Defines the name of the **org.picketlink. identity.federation. core.interfaces. RoleGenerator** subclass to be used to obtain user roles.	Defaults to **org.picketlink.identity. federation.core. impl.EmptyRoleGenerator** .
AttributeManager	Defines the name of the **org.picketlink. identity.federation. core.interfaces. AttributeManager** subclass to be used to obtain the SAML assertion attributes.	Defautls to **org.picketlink. identity.federation. core.impl. EmptyAttributeManager** .
StrictPostBinding	SAML Web Browser SSO Profile has a requirement that the IDP does not respond back in Redirect Binding. Set this to false if you want to force the IDP to respond to SPs using the Redirect Binding.	Values: **true|false** . Defaults to true, the IDP always respond via POST Binding.
SupportsSignatures	Indicates if digital signature/ verification of SAML assertions are enabled. If this attribute is marked to true the Service Providers must support signatures too, otherwise the SAML messages will be considered as invalid.	Values: **true|false.** Defaults to false.
Encrypt	Indicates if SAML Assertions should be encrypted. If this attribute is marked to true the Service Providers must support signatures too, otherwise the SAML messages will be considered as invalid.	Values: **true|false.** Defaults to false
IdentityParticipantStack	Defines the name of the **org.picketlink. identity.federation. web.core. IdentityParticipantStack** subclass to be used to register and deregister participants in the identity federation.	Defaults to **org.picketlink. identity.federation. web.core. IdentityServer.STACK.**

Report a bug[25]

[25] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29792-620398+%5BLatest

14.5.3.4.3.1.1. IdentityURL Element

This element value refers to the URL of the Identity Provider.

Eg.: *http://localhost:8080/idp/*

Report a bug[26]

14.5.3.4.3.1.2. Trust/Domains Elements

The Trust and Domains elements defines the hosts trusted by this Identity Provider. You just need to inform a list of comma separated domain names.

Report a bug[27]

14.5.3.4.3.1.3. SAML Digital Signature Configuration (KeyProvider Element)

To enable digital signatures for the SAML assertions you need to configure:

1. Set the **SupportsSignature** attribute to true;
2. Add the *Section 14.5.7.11, "SAML2SignatureGenerationHandler"* and the *Section 14.5.7.12, "SAML2SignatureValidationHandler"* in the handlers chain (Handler Element).
3. Configure a ***Section 14.5.6, "Digital Signatures in SAML Assertions"*** * *element.

Report a bug[28]

14.5.3.4.3.1.4. SAML Encryption Configuration

To enable encryption for SAML assertions you need to configure:

1. Set the **Encrypt** attribute to true;
2. Add the ***Section 14.5.7.8, "SAML2EncryptionHandler"*** and the *Section 14.5.7.12, "SAML2SignatureValidationHandler"* in the handlers chain (Handler Element).
3. Configure a ***Section 14.5.6, "Digital Signatures in SAML Assertions"*** * *element.

Report a bug[29]

%5D&comment=Title%3A+PicketLinkIDP+Element%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[26] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29793-626783+%5BLatest%5D&comment=Title%3A+IdentityURL+Element%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29793-626783+01+Apr+2014+21%3A00+en-US+%5BLatest%5D

[27] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29794-620400+%5BLatest%5D&comment=Title%3A+Trust%2FDomains+Elements%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29794-620400+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[28] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29795-620678+%5BLatest%5D&comment=Title%3A+SAML+Digital+Signature+Configuration+%28KeyProvider+Element%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29795-620678+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[29] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29796-620679+%5BLatest%5D&comment=Title%3A+SAML+Encryption+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for

14.5.3.4.3.2. SAML Handlers Configuration (Handlers Element)

PicketLink provides some built-in *Section 14.5.7, "SAML2 Handlers"* to help the Identity Provider Authenticator processing the SAML requests and responses.

The handlers are configured through the **Handlers** element.

Report a bug[30]

14.5.3.4.3.3. SecurityToken Service Configuration (PicketLinkSTS Element)

Important

When configuring the IDP, you do not need to specify the PicketLinkSTS element in the configuration. If it is ommited PicketLink will load the default configurations from a file named core-sts inside the picketlink-core-VERSION.jar.

Override this configuration only if you need to. Eg.: change the token timeout or specify a custom Security Token Provider for SAML assertions.

See the documentation at *Section 14.5.3.6, "Security Token Service Configuration"* .

Report a bug[31]

14.5.3.5. Identity Stores

14.5.3.5.1. Introduction

The Identity Provider needs a Identity Store to retrieve users information. These informations will be used during the authentication and authorization process. Identity Stores can be any type of repository: a database, LDAP, properties file, etc.

The PicketLink Identity Provider uses JAAS to connect to an Identity Store. This configuration is usually made at the container side using any LoginModule implementation.

If you are using the JBoss Application Server you can use one of the existing LoginModules or you can create your custom implementation:

- *https://community.jboss.org/wiki/JBossAS7SecurityDomainModel*

+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29796-620679+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[30] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29797-620680+%5BLatest%5D&comment=Title%3A+SAML+Handlers+Configuration+%28Handlers+Element%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29797-620680+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[31] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29798-620681+%5BLatest%5D&comment=Title%3A+SecurityToken+Service+Configuration+%28PicketLinkSTS+Element%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29798-620681+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

Report a bug[32]

14.5.3.5.2. Configuring a Security Domain for a Identity Store

In order to authenticate users, the Identity Provider needs to be configured with the properly security domain configuration. The security domain is responsible for authenticating the user in a specific Identity Store.

This is done by defining a **<security-domain>** element in jboss-web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
    <security-domain>idp</security-domain>
    <valve>
  <class-name>org.picketlink.identity.federation.bindings.tomcat.idp.IDPWebBrowserSSOValve</
class-name>
    </valve>
</jboss-web>
```

In order to use the security domain above, you need to configure it in your server. For JBoss AS7 you just need to add the following configuration to standalone.xml:

```
<subsystem xmlns="urn:jboss:domain:security:1.1">
    <security-domains>
        <security-domain name="idp" cache-type="default">
            <authentication>
                <login-module code="UsersRoles" flag="required">
                    <module-option name="usersProperties" value="users.properties"/>
                    <module-option name="rolesProperties" value="roles.properties"/>
                </login-module>
            </authentication>
    </security-domain>

    ...

</subsystem>
```

The example above uses a JAAS LoginModule that uses two properties files to authenticate users and retrieve their roles. These properties files needs to be located at WEB-INF/classes folder.

Report a bug[33]

14.5.3.6. Security Token Service Configuration

14.5.3.6.1. SecurityToken Service Configuration (PicketLinkSTS Element)

To issue/renew/cancel/validate SAML tokens, the IDP relies on the PicketLink STS API and configuration. This configurations define how the tokens should be used by the IDP.

[32] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29799-626784+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29799-626784+01+Apr+2014+21%3A01+en-US+%5BLatest%5D

[33] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29800-620406+%5BLatest%5D&comment=Title%3A+Configuring+a+Security+Domain+for+a+Identity+Store%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29800-620406+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

This *PicketLinkSTS* element defines the basic configuration for the Security Token Service. The table bellow provides more information about the attributes supported by this element:

Name	Description	Value
STSName	Name for this STS configuration.	Name for this Security Token Service.
TokenTimeout	Defines the token timeout in miliseconds.	Defaults to **3600** miliseconds.
ClockSkew	Defines the clock skew, or timing skew, for the token timeout.	Defaults to **2000** miliseconds.
SignToken	Indicates if the tokens should be signed.	Values: **true|false** . Defaults to **false** .
EncryptToken	Indicates if the tokens should be encrypted.	Values: **true|false** . Defaults to **false** .
CanonicalizationMethod	Sets the canonicalization method.	Defaults to *http://www.w3.org/2001/10/xml-exc-c14n#WithComments*

Report a bug[34]

14.5.3.6.1.1. Security Token Providers (TokenProviders/TokenProvider elements)

The PicketLink STS defines the concept of *Security Token Providers.* This tokens providers are implementations of the interface **org.picketlink.identity.federation.core.interfaces.SecurityTokenProvider** .

The purpose of providers is to plug any implementation for a specific token type. PicketLink provides default implementations for the following token type:

- **SAML v2.0** : *org.picketlink.identity.federation.core.saml.v2.providers.SAML20AssertionTokenProvider*
- **WS-Trust _** : org.picketlink.identity.federation.core.wstrust.plugins.saml.SAML20TokenProvider_

Each provider is linked to a specific *TokenType* and *TokenElementNS,* both attributes of the *TokenProvider* element.

You can always provide your own implementation for a specific TokenType or customize the behaviour for one of the built-in providers.

Report a bug[35]

[34] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29801-626779+%5BLatest%5D&comment=Title%3A+SecurityToken+Service+Configuration+%28PicketLinkSTS+Element%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[35] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29802-620408+%5BLatest%5D&comment=Title%3A+Security+Token+Providers+%28TokenProviders%2FTokenProvider+elements%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29802-620408+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

14.5.4. PicketLink Service Provider (PSP)

14.5.4.1. Introduction

The PicketLink Service Provider relies on the PicketLink Identity Provider to assert information about a user via an electronic user credential, leaving the service provider to manage access control and dissemination based on a trusted set of user credential assertions.

Tip

Please have a look at the *PicketLink Quickstarts*[36] to obtain service provider applications. The quickstarts are useful resources where you can get configuration files.

Report a bug[37]

14.5.4.2. How to create your own PicketLink Service Provider

The best way to create your own Service Provider implementation is using one of the examples provided by the PicketLink Quickstarts.

You should also take a look at the following documentations:

- *Section 14.5.4.3, "Service Provider Configuration"*
- *Section 14.5.4.4, "Service Provider Authenticators"*
- *Section 14.5.4.5, "Service Provider Security Domain"*

Report a bug[38]

14.5.4.3. Service Provider Configuration

14.5.4.3.1. Configuring a Service Provider

To configure an application as a PicketLink Service Provider you need to follow this steps:

1. Configuring the web.xml.
2. Configure an *Section 14.5.4.4, "Service Provider Authenticators"* .
3. Configure a *Section 14.5.4.5, "Service Provider Security Domain"* for your application.

[36] https://docs.jboss.org/author/pages/viewpage.action?pageId=23986289
[37] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29803-626780+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29803-626780+01+Apr+2014+20%3A59+en-US+%5BLatest%5D
[38] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29804-620682+%5BLatest%5D&comment=Title%3A+How+to+create+your+own+PicketLink+Service+Provider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29804-620682+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

4. Configure ***PicketLink JBoss Module*** [39] as a dependency.

5. Create and configure a file named **WEB-INF/picketlink.xml** .

Report a bug[40]

14.5.4.3.2. Configuring the web.xml

Before configuring your application as an Service Provider you need to add some configurations to your web.xml.

Let's start by defining a **security-constraint** element to restrict access to resources from unauthenticated users:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Manager command</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>manager</role-name>
    </auth-constraint>
</security-constraint>

<security-role>
    <description>
      The role that is required to log in to the Manager Application
    </description>
    <role-name>manager</role-name>
</security-role>
```

As you can see above, we define that only users with a role named **manager** are allowed to access the protected resources. Make sure to give your users the same role you defined here, otherwise they will get a 403 HTTP status code.

During the logout process, PicketLink will try to redirect the user to a **logout.jsp** page located at the root directory of your application. Please, make sure to create it.

Important

Please, make sure you have a welcome file page in your application. You can define it in your web.xml or simply create an **index.jsp** at the root directory of your application.

Report a bug[41]

[39] https://docs.jboss.org/author/display/PLINK/JBoss+Modules
[40] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29805-626781+%5BLatest%5D&comment=Title%3A+Configuring+a+Service+Provider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29805-626781+01+Apr+2014+21%3A00+en-US+%5BLatest%5D
[41] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29806-620412+%5BLatest%5D&comment=Title%3A+Configuring+the+web.xml%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for

14.5.4.3.3. The picketlink.xml configuration file

All the configuration for an especific Service Providers goes at the WEB-INF/picketlink.xml file. This file is responsible to define the behaviour of the Authenticator. During the service provider startup, the authenticator parses this file and configures itself.

Bellow is how the picketlink.xml file should looks like:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">


    <PicketLinkSP xmlns="urn:picketlink:identity-federation:config:2.1"
        BindingType="REDIRECT"
        RelayState="someURL"
        ErrorPage="/someerror.jsp"
        LogOutPage="/customLogout.jsp"
        IDPUsesPostBinding="true"
        SupportsSignatures="true">

        <IdentityURL>http://localhost:8080/idp/ </IdentityURL>
        <ServiceURL>http://localhost:8080/employee/ </ServiceURL>

        <KeyProvider
 ClassName="org.picketlink.identity.federation.core.impl.KeyStoreKeyManager">

            <Auth Key="KeyStoreURL" Value="/jbid_test_keystore.jks" />
            <Auth Key="KeyStorePass" Value="store123" />
            <Auth Key="SigningKeyPass" Value="test123" />
            <Auth Key="SigningKeyAlias" Value="servercert" />

            <ValidatingAlias Key="localhost" Value="servercert" />
            <ValidatingAlias Key="127.0.0.1" Value="servercert" />

        </KeyProvider>

    </PicketLinkSP>

    <Handlers xmlns="urn:picketlink:identity-federation:handler:config:2.1">

        <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler" />
        <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler" />
        <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler" />
        <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler" />

     </Handlers>

</PicketLink>
```

+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29806-620412+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

Important

The schema for the picketlink.xml file is available here: *https://github.com/picketlink/picketlink/blob/master/modules/federation/src/main/resources/schema/config/picketlink_v2.1.xsd* .

Report a bug[42]

14.5.4.3.3.1. PicketLinkSP Element

This element defines the basic configuration for the service provider. The table bellow provides more information about the attributes supported by this element:

Name	Description	Value
BindingType	Defines which SAML binding should be used: SAML HTTP POST or Redirect bindings.	POST|REDIRECT. Defaults to POST if no specified.
ErrorPage	Defines a custom error page to be displayed when some error occurs during the request processing.	Defaults to /error.jsp.
LogOutPage	Defines a custom logout page to be displayed after the logout.	Defaults to /logout.jsp.
IDPUsesPostBinding	Indicates if the Identity Provider configured for this Service Provider is always using POST for SAML responses.	true|false. Defaults to true if no specified.
SupportsSignatures	Indicates if digital signature/ verification of SAML assertions are enabled. If this attribute is marked to true the Identity Provider configured for this Service Provider must support signatures too, otherwise the SAML messages will be considered as invalid.	true|false. Defaults to false if no specified.

Report a bug[43]

[42] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29807-682437+%5BLatest%5D&comment=Title%3A+The+picketlink.xml+configuration+file%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[43] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29808-620414+%5BLatest%5D&comment=Title%3A+PicketLinkSP+Element%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.5.4.3.3.1.1. IdentityURL Element

This element value refers to the URL of the Identity Provider used by this Service Provider.

Eg.: *http://localhost:8080/idp/*

Report a bug[44]

14.5.4.3.3.1.2. ServiceURL Element

This element value refers to the URL of the Service Provider.

Eg.: *http://localhost:8080/sales/*

Report a bug[45]

14.5.4.3.3.2. SAML Digital Signature Configuration (KeyProvider Element)

To enable digital signatures for the SAML assertions you need to configure:

1. Set the **SupportsSignature** attribute to true;
2. Add the *Section 14.5.7.11, "SAML2SignatureGenerationHandler"* and the *Section 14.5.7.12, "SAML2SignatureValidationHandler"* in the handlers chain (Handler Element).
3. Configure a ***Section 14.5.6, "Digital Signatures in SAML Assertions"*** * *element.

Report a bug[46]

14.5.4.3.3.3. SAML Handlers Configuration (Handlers Element)

PicketLink provides some built-in *Section 14.5.7, "SAML2 Handlers"* to help the Service Provider Authenticator processing the SAML requests and responses.

The handlers are configured through the **Handlers** element.

Report a bug[47]

[44] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29809-626786+%5BLatest%5D&comment=Title%3A+IdentityURL+Element%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29809-626786+01+Apr+2014+21%3A02+en-US+%5BLatest%5D

[45] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29810-626787+%5BLatest%5D&comment=Title%3A+ServiceURL+Element%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29810-626787+01+Apr+2014+21%3A02+en-US+%5BLatest%5D

[46] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29811-620684+%5BLatest%5D&comment=Title%3A+SAML+Digital+Signature+Configuration+%28KeyProvider+Element%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29811-620684+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[47] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29812-620685+%5BLatest%5D&comment=Title%3A+SAML+Handlers+Configuration+%28Handlers+Element%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29812-620685+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

14.5.4.4. Service Provider Authenticators

14.5.4.4.1. Introduction

Error

Topic 29813 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 29813* for more detailed information.

14.5.4.4.2. Configuring an Authenthicator for a Service Provider

The PicketLink Authenticator is basically a *Tomcat Valve*[48] (org.apache.catalina.authenticator.FormAuthenticator). The only thing you need to do is change the valves configuration for your application.

This configuration changes for each supported binding.

Report a bug[49]

14.5.4.4.2.1. JBoss Application Server v7

In JBoss Application Server v7 the valves configuration are located inside the **WEB-INF/jboss-web.xml** file. Bellow is a example of how this file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
 <security-domain>sp</security-domain>
 <context-root>employee</context-root>
 <valve>
     <class-
name>org.picketlink.identity.federation.bindings.tomcat.sp.ServiceProviderAuthenticator</
class-name>
 </valve>
</jboss-web>
```

The valve configuration is done using the **<valve>** element.

Report a bug[50]

14.5.4.4.2.2. JBoss Application Server v5 or v6

In JBoss Application Server v5 or v6, the valves configuration are located inside the **WEB-INF/context.xml** file. Bellow is a example of how this file looks like:

[48] http://tomcat.apache.org/tomcat-6.0-doc/config/valve.html
[49] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29814-626788+%5BLatest%5D&comment=Title%3A+Configuring+an+Authenthicator+for+a+Service+Provider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A
[50] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29815-620421+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v7%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29815-620421+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Valve
 className="org.picketlink.identity.federation.bindings.tomcat.sp.ServiceProviderAuthenticator" /
>
</Context>
```

The valve configuration is done using the **<Valve>** element.

Report a bug[51]

14.5.4.4.2.3. Apache Tomcat 6

In Apache Tomcat 6 the valves configuration are located inside the **META-INF/context.xml** file. Bellow is a example of how this file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Valve
 className="org.picketlink.identity.federation.bindings.tomcat.sp.ServiceProviderAuthenticator" /
>
</Context>
```

The valve configuration is done using the **<Valve>** element.

Report a bug[52]

14.5.4.4.3. Built-in Authenticators

PicketLink provides default implementations for Service Provider Authenticators. The list bellow shows all the available implementations:

Name	Description
Section 14.5.4.4.4, "ServiceProviderAuthenticator"	Preferred service provider authenticator. Supports both SAML HTTP Redirect and POST bindings.
Section 14.5.4.4.8, "SPPostFormAuthenticator"	**Deprecated** . Supports only HTTP POST Binding without signature of SAML assertions.
Section 14.5.4.4.7, "SPPostSignatureFormAuthenticator"	**Deprecated** . Supports only HTTP POST Binding with signature of SAML assertions.
Section 14.5.4.4.6, "SPRedirectFormAuthenticator"	**Deprecated** . Supports only HTTP Redirect Binding without signature of SAML assertions.
Section 14.5.4.4.5, "SPRedirectSignatureFormAuthenticator"	**Deprecated** . Supports only HTTP Redirect Binding with signature of SAML assertions.

[51] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29816-620422+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v5+or+v6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29816-620422+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[52] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29817-620423+%5BLatest%5D&comment=Title%3A+Apache+Tomcat+6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29817-620423+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

Warning

Prefer using the *Section 14.5.4.4.4, "ServiceProviderAuthenticator"* ServiceProviderAuthenticator authenticator if you are using PicketLink v.2.1 or above. The others authenticators are **DEPRECATED** .

Report a bug[53]

14.5.4.4.4. ServiceProviderAuthenticator

As of PicketLink v2.1, the ServiceProviderAuthenticator is the preferred Service Provider configuration to the deprecated *Section 14.5.4.4.8, "SPPostFormAuthenticator"* , *Section 14.5.4.4.6, "SPRedirectFormAuthenticator"* , *Section 14.5.4.4.7, "SPPostSignatureFormAuthenticator"* and *Section 14.5.4.4.5, "SPRedirectSignatureFormAuthenticator"* .

Report a bug[54]

14.5.4.4.4.1. Configuration

https://docs.jboss.org/author/display/PLINK/Service+Provider+Configuration

Report a bug[55]

14.5.4.4.5. SPRedirectSignatureFormAuthenticator

Warning

As of PicketLink v2.1, the *Section 14.5.4.4.4, "ServiceProviderAuthenticator"* is the preferred Service Provider configuration to the **deprecated** *Section 14.5.4.4.8, "SPPostFormAuthenticator"* , *Section 14.5.4.4.6, "SPRedirectFormAuthenticator"* , *Section 14.5.4.4.7, "SPPostSignatureFormAuthenticator"* and *Section 14.5.4.4.5, "SPRedirectSignatureFormAuthenticator"* .

[53] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29818-620686+%5BLatest%5D&comment=Title%3A+Built-in+Authenticators%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29818-620686+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[54] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29819-620687+%5BLatest%5D&comment=Title%3A+ServiceProviderAuthenticator%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[55] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29820-626785+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29820-626785+01+Apr+2014+21%3A01+en-US+%5BLatest%5D

SPRedirectSignatureFormAuthenticator is used to provide signature/encryption services to a Service Provider (SP) application for HTTP/Redirect binding of SAMLv2 specification. This authenticator

is an extension of the *Section 14.5.4.4.6, "SPRedirectFormAuthenticator"* .

Report a bug[56]

14.5.4.4.5.1. Binding

HTTP/Redirect Binding (along with signature/encryption support)

Report a bug[57]

14.5.4.4.5.2. Configuration

14.5.4.4.5.2.1. JBoss Application Server v5.x/6

Configure in WEB-INF/context.xml

Report a bug[58]

14.5.4.4.5.2.2. Apache Tomcat v5.5/6.x

Configure in META-INF/context.xml

Report a bug[59]

14.5.4.4.5.2.3. Example:

Example 14.2. context.xml

```
<Context>
  <Valve
 className="org.picketlink.identity.federation.bindings.tomcat.sp.SPRedirectSignatureFormAuthenticator
  />
</Context>
```

[56] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29822-620688+%5BLatest%5D&comment=Title%3A+SPRedirectSignatureFormAuthenticator%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[57] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29823-620429+%5BLatest%5D&comment=Title%3A+Binding%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29823-620429+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[58] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29824-620430+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v5.x%2F6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29824-620430+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[59] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29825-620431+%5BLatest%5D&comment=Title%3A+Apache+Tomcat+v5.5%2F6.x%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29825-620431+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

Report a bug[60]

14.5.4.4.5.2.4. Attributes

#	Name	Type	Objective	Since
1	configFile	String	optional - fully qualified location of the config file Default: /WEB-INF/picketlink-idfed.xml	2.0
2	samlHandlerChainClass	String	optional - fqn of a custom SAMLHandlerChain implementation	2.0
3	serviceURL	String	optional - the service provider URL	2.0
4	saveRestoreRequest	boolean	should the authenticator save the original request and restore it after authentication Default: true	2.0
5	configProvider	String	optional - a fqn of the SAMLConfigurationProvider implementation	2.0
6	issuerID	String	optional - customize the issuer id	2.0
7	idpAddress	String	optional - If the request.getRemoteAddr is not exactly the IDP address that you have keyed in your deployment descriptor for keystore alias, you can configure it explicitly	2.0

Report a bug[61]

14.5.4.4.6. SPRedirectFormAuthenticator

Warning

As of PicketLink v2.1, the *Section 14.5.4.4.4, "ServiceProviderAuthenticator"* is the preferred Service Provider configuration to the **deprecated** *Section 14.5.4.4.8, "SPPostFormAuthenticator"* , *Section 14.5.4.4.6, "SPRedirectFormAuthenticator"* , *Section 14.5.4.4.7, "SPPostSignatureFormAuthenticator"* and *Section 14.5.4.4.5, "SPRedirectSignatureFormAuthenticator"* .

SPRedirectFormAuthenticator provides the SAMLv2 HTTP/Redirect binding support for service provider (SP) applications.

[60] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29827-620433+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29827-620433+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[61] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29828-620434+%5BLatest%5D&comment=Title%3A+Attributes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29828-620434+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

Report a bug[62]

14.5.4.4.6.1. Binding

SAMLv2 HTTP/Redirect Binding

Report a bug[63]

14.5.4.4.6.2. Configuration

14.5.4.4.6.2.1. JBoss Application Server v5.x/6

Configure in WEB-INF/context.xml

Report a bug[64]

14.5.4.4.6.2.2. Apache Tomcat v5.5/6.x

Configure in META-INF/context.xml

Report a bug[65]

14.5.4.4.6.2.3. Example:

Example 14.3. context.xml

```
<Context>
  <Valve
 className="org.picketlink.identity.federation.bindings.tomcat.sp.SPRedirectFormAuthenticator"
  />
</Context>
```

Report a bug[66]

[62] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29829-620689+%5BLatest%5D&comment=Title%3A+SPRedirectFormAuthenticator%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[63] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29830-620436+%5BLatest%5D&comment=Title%3A+Binding%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29830-620436+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[64] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29831-620437+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v5.x%2F6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29831-620437+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[65] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29832-620438+%5BLatest%5D&comment=Title%3A+Apache+Tomcat+v5.5%2F6.x%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29832-620438+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[66] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29834-620440+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29834-620440+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

14.5.4.4.6.2.4. Attributes

#	Name	Type	Objective	Since
1	configFile	String	optional - fully qualified location of the config file Default: /WEB-INF/picketlink-idfed.xml	2.0
2	samlHandlerChainClass	String	optional - fqn of a custom SAMLHandlerChain implementation	2.0
3	serviceURL	String	optional - the service provider URL	2.0
4	saveRestoreRequest	boolean	should the authenticator save the original request and restore it after authentication Default: true	2.0
5	configProvider	String	optional - a fqn of the SAMLConfigurationProvider implementation	2.0
6	issuerID	String	optional - customize the issuer id	2.0

Report a bug[67]

14.5.4.4.7. SPPostSignatureFormAuthenticator

Warning

As of PicketLink v2.1, the *Section 14.5.4.4.4, "ServiceProviderAuthenticator"* is the preferred Service Provider configuration to the **deprecated** *Section 14.5.4.4.8, "SPPostFormAuthenticator"* , *Section 14.5.4.4.6, "SPRedirectFormAuthenticator"* , *Section 14.5.4.4.7, "SPPostSignatureFormAuthenticator"* and *Section 14.5.4.4.5, "SPRedirectSignatureFormAuthenticator"* .

SPPostSignatureFormAuthenticator is used to provide signature/encryption services to a Service Provider (SP) application for HTTP/POST binding of SAMLv2 specification. This authenticator

is an extension of the *Section 14.5.4.4.8, "SPPostFormAuthenticator"* .

Report a bug[68]

14.5.4.4.7.1. Binding

HTTP/POST Binding (along with signature/encryption support)

[67] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29835-620441+%5BLatest%5D&comment=Title%3A+Attributes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29835-620441+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[68] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29836-620690+%5BLatest%5D&comment=Title%3A+SPPostSignatureFormAuthenticator%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

Report a bug[69]

14.5.4.4.7.2. Configuration

14.5.4.4.7.2.1. JBoss Application Server v5.x/6

Configure in WEB-INF/context.xml

Report a bug[70]

14.5.4.4.7.2.2. Apache Tomcat v5.5/6.x

Configure in META-INF/context.xml

Report a bug[71]

14.5.4.4.7.2.3. Example:

Example 14.4. context.xml

```
<Context>
  <Valve
 className="org.picketlink.identity.federation.bindings.tomcat.sp.SPPostSignatureFormAuthenticator"
  />
</Context>
```

Report a bug[72]

14.5.4.4.7.2.4. Attributes

#	Name	Type	Objective	Since
1	configFile	String	optional - fully qualified location of the config file Default: /WEB-INF/picketlink-idfed.xml	2.0
2	samlHandlerChainClass	String	optional - fqn of a custom SAMLHandlerChain implementation	2.0

[69] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29837-620443+%5BLatest%5D&comment=Title%3A+Binding%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29837-620443+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[70] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29838-620444+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v5.x%2F6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29838-620444+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[71] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29839-620445+%5BLatest%5D&comment=Title%3A+Apache+Tomcat+v5.5%2F6.x%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29839-620445+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[72] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29841-620447+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29841-620447+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

#	Name	Type	Objective	Since
3	serviceURL	String	optional - the service provider URL	2.0
4	saveRestoreRequest	boolean	should the authenticator save the original request and restore it after authentication Default: true	2.0
5	configProvider	String	optional - a fqn of the SAMLConfigurationProvider implementation	2.0
6	issuerID	String	optional - customize the issuer id	2.0
7	idpAddress	String	optional - If the request.getRemoteAddr is not exactly the IDP address that you have keyed in your deployment descriptor for keystore alias, you can configure it explicitly	2.0

Report a bug[73]

14.5.4.4.8. SPPostFormAuthenticator

Warning

As of PicketLink v2.1, the *Section 14.5.4.4.4, "ServiceProviderAuthenticator"* is the preferred Service Provider configuration to the **deprecated** *Section 14.5.4.4.8, "SPPostFormAuthenticator"* , *Section 14.5.4.4.6, "SPRedirectFormAuthenticator"* , *Section 14.5.4.4.7, "SPPostSignatureFormAuthenticator"* and *Section 14.5.4.4.5, "SPRedirectSignatureFormAuthenticator"* .

SPPostFormAuthenticator is the main authenticator used to configure a service provider (SP) application for SAMLv2.0

Report a bug[74]

14.5.4.4.8.1. Binding

SAMLv2 HTTP/Post Binding

Report a bug[75]

[73] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29842-620448+%5BLatest%5D&comment=Title%3A+Attributes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29842-620448+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[74] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29843-620691+%5BLatest%5D&comment=Title%3A+SPPostFormAuthenticator%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[75] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29844-620450+%5BLatest

14.5.4.4.8.2. Configuration

14.5.4.4.8.2.1. JBoss Application Server v5.x/6

Configure in WEB-INF/context.xml

Report a bug[76]

14.5.4.4.8.2.2. Apache Tomcat v5.5/6.x

Configure in META-INF/context.xml

Report a bug[77]

14.5.4.4.8.2.3. Example:

Example 14.5. context.xml

```
<Context>
  <Valve
 className="org.picketlink.identity.federation.bindings.tomcat.sp.SPPostFormAuthenticator"
  />
</Context>
```

Report a bug[78]

14.5.4.4.8.2.4. Attributes

#	Name	Type	Objective	Since
1	configFile	String	optional - fully qualified location of the config file Default: /WEB-INF/picketlink-idfed.xml	2.0
2	samlHandlerChainClass	String	optional - fqn of a custom SAMLHandlerChain implementation	2.0
3	serviceURL	String	optional - the service provider URL	2.0
4	saveRestoreRequest	boolean	should the authenticator save the original request and restore it after authentication Default: true	2.0

%5D&comment=Title%3A+Binding%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29844-620450+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[76] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29845-620451+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+v5.x%2F6%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29845-620451+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[77] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29846-620452+%5BLatest%5D&comment=Title%3A+Apache+Tomcat+v5.5%2F6.x%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29846-620452+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[78] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29848-620454+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29848-620454+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

#	Name	Type	Objective	Since
5	configProvider	String	optional - a fqn of the SAMLConfigurationProvider implementation	2.0
6	issuerID	String	optional - customize the issuer id	2.0

Report a bug[79]

14.5.4.5. Service Provider Security Domain

14.5.4.5.1. Configuring a security domain

In order to handle the SAML assertions returned by the Identity Provider, the Service Provider needs to be configured with the properly security domain configuration. This is done by defining a **<security-domain>** element in jboss-web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
    <security-domain>sp</security-domain>
    <valve>
        <class-
name>org.picketlink.identity.federation.bindings.tomcat.sp.ServiceProviderAuthenticator</
class-name>
    </valve>
</jboss-web>
```

In order to use the security domain above, you need to configure it in your server. For JBoss AS7 you just need to add the following configuration to standalone.xml:

```
<subsystem xmlns="urn:jboss:domain:security:1.1">
    <security-domains>
        <security-domain name="sp" cache-type="default">
            <authentication>
                <login-module
 code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2LoginModule"
 flag="required"/>
            </authentication>
    </security-domain>

    ...

</subsystem>
```

Report a bug[80]

[79] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29849-620455+%5BLatest%5D&comment=Title%3A+Attributes%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29849-620455+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[80] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29850-620456+%5BLatest%5D&comment=Title%3A+Configuring+a+security+domain%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29850-620456+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

14.5.5. SAML Authenticators (Tomcat,JBossAS)

14.5.5.1. Introduction

Error

Topic 29851 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 29851* for more detailed information.

14.5.5.2. Tomcat Authenticators for use in Apache Tomcat and JBoss Application Server

PicketLink includes a number of Authenticators for providing SAML support on Apache Tomcat and JBoss Application Server.

Report a bug[81]

14.5.5.2.1. Authenticators/Valves for Identity Provider (IDP)

1. *Section 14.5.3.3.4, "IDPWebBrowserSSOValve"*

Report a bug[82]

14.5.5.2.2. Authenticators/Valves for Service Provider (SP)

14.5.5.2.2.1. Deprecated (as of PicketLink v2.1)

1. *Section 14.5.4.4.8, "SPPostFormAuthenticator"*
2. *Section 14.5.4.4.6, "SPRedirectFormAuthenticator"*
3. *Section 14.5.4.4.7, "SPPostSignatureFormAuthenticator"*
4. *Section 14.5.4.4.5, "SPRedirectSignatureFormAuthenticator"*

Report a bug[83]

[81] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29852-620458+%5BLatest%5D&comment=Title%3A+Tomcat+Authenticators+for+use+in+Apache+Tomcat+and+JBoss+Application+Server%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[82] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29853-620692+%5BLatest%5D&comment=Title%3A+Authenticators%2FValves+for+Identity+Provider+%28IDP%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29853-620692+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[83] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29854-620693+%5BLatest%5D&comment=Title%3A+Deprecated+%28as+of+PicketLink+v2.1%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29854-620693+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

14.5.5.3. Useful Information

- *Tomcat Character Encoding (UTF-8 etc)*[84]

Report a bug[85]

14.5.6. Digital Signatures in SAML Assertions

14.5.6.1. Configuring the KeyProvider

To support digital signatures of SAML assertions you should define a KeyProvider element inside a PicketLinkIDP or PicketLinkSP.

Important

When using digital signatures you need to configure and enable it in both Identity Provider and Service Providers. Otherwise the SAML assertions would probably be considered as invalid.

```
<KeyProvider ClassName="org.picketlink.identity.federation.core.impl.KeyStoreKeyManager">
    <Auth Key="KeyStoreURL" Value="/jbid_test_keystore.jks" />
    <Auth Key="KeyStorePass" Value="store123" />
    <Auth Key="SigningKeyPass" Value="test123" />
    <Auth Key="SigningKeyAlias" Value="servercert" />

    <ValidatingAlias Key="idp.example.com" Value="servercert" />
    <ValidatingAlias Key="localhost" Value="servercert" />
</KeyProvider>
```

In order to configure the KeyProvider, you need to specify some configurations about the Java KeyStore that should be used to sign SAML assertions:

Auth Key	Description
KeyStoreURL	Where the value of the **Value** attribute points to the location of a Java KeyStore with the properly installed certificates.
KeyStorePass	Where the value of the **Value** attribute refers to the password of the referenced Java KeyStore.
SigningKeyAlias	Where the value of the **Value** attribute refers to the password of the installed certificate to be used to sign the SAML assertions.
SigningKeyPass	Where the value of the **Value** attribute refers to the alias of the certificate to be used to sign the SAML assertions.

[84] http://wiki.apache.org/tomcat/FAQ/CharacterEncoding
[85] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29856-626722+%5BLatest%5D&comment=Title%3A+Useful+Information%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29856-626722+01+Apr+2014+16%3A22+en-US+%5BLatest%5D

The Service Provider also needs to know how to verify the signatures for the SAML assertions. This is done by the **ValidationAlias** elements.

```
<ValidatingAlias Key="idp.example.com" Value="servercert" />
```

Tip

Note that we declare the validating certificate for each domain using the *ValidatingAlias* .

At the IDP side you need an entry for each server/domain name defined as a trusted domain (Trust/Domains elements).

At the SP side you need an entry for the the server/domain name where the IDP is deployed.

Report a bug[86]

14.5.6.2. SImple Example Scenario

14.5.6.2.1. How SAML assertions are signed ?

When digital signatures are enable, the authenticator will look at the **SigningKeyAlias** for the alias that should me used to look for a private key configured in the Java KeyStore. This private key will be used to sign the SAML assertion.

Report a bug[87]

14.5.6.2.2. How signatures are validated ?

When digital signatures are enabled, the authenticator will look at the ValidatingAlias table for a entry that matches the value of the **Key** attribute with the host name of the Issuer of the SAML assertion. For example, consider the following SAML Assertion issued by an Identity Provider located at *http://idp.example.com*

```
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    ID="ID_ab0392ef-b557-4453-95a8-a7e168da8ac5" IssueInstant="2010-09-30T19:13:37.869Z"
    Version="2.0">
    <saml2:Issuer>http://idp.example.com </saml2:Issuer>
    <saml2:Subject>
        <saml2:NameID NameQualifier="urn:picketlink:identity-federation">jduke</saml2:NameID>
        <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer" />
    </saml2:Subject>
    <saml2:Conditions NotBefore="2010-09-30T19:13:37.869Z"
```

[86] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29857-620463+%5BLatest%5D&comment=Title%3A+Configuring+the+KeyProvider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29857-620463+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[87] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29858-620464+%5BLatest%5D&comment=Title%3A+How+SAML+assertions+are+signed+%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29858-620464+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

```
         NotOnOrAfter="2010-09-30T21:13:37.869Z" />
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#WithComments" />
            <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlds#rsa-sha1" />
            <ds:Reference URI="#ID_ab0392ef-b557-4453-95a8-a7e168da8ac5">
                <ds:Transforms>
                    <ds:Transform Algorithm="http://www.w3.org/2000/09/xmlds#enveloped-
signature" />
                    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                </ds:Transforms>
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#sha1" />
                <ds:DigestValue>0Y9QM5c5qCShz5UWmbFzBmbuTus=</ds:DigestValue>
            </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
            se/flQ2htUQ0IUYieVkXNn9cfjnfgv6H99nFarsTNTpRI9xuSlw5OTai/2PYdZI2Va9+QzzBf99m
            VFyigfFdfrqug6aKFhF0lsujzlFfPfmXBbDRiTFX+4SkBeV71uuy7rOUI/jRiitEA0QrKqs0e/pV
            \+C8PoaariisK96Mtt7A=
        </ds:SignatureValue>
        <ds:KeyInfo>
            <ds:KeyValue>
                <ds:RSAKeyValue>
                    <ds:Modulus>
                        suGIyhVTbFvDwZdx8Av62zmP
+aGOlsBN8WUE3eEEcDtOIZgO78SImMQGwB2C0eIVMhiLRzVPqoW1
                        dCPAveTm653zHOmubaps1fY0lLJDSZbTbhjeYhoQmmaBro/
tDpVw5lKJwspqVnMuRK19ju2dxpKw
                        lYGGtrP5VQv00dfNPbs=
                    </ds:Modulus>
                    <ds:Exponent>AQAB</ds:Exponent>
                </ds:RSAKeyValue>
            </ds:KeyValue>
        </ds:KeyInfo>
    </ds:Signature>
</saml2:Assertion>
```

During the signature validation for this SAML assertion, the authenticator (in this case a Service Provider Authenticator) will try to find a **ValidationAlias** element with the value **idp.example.com** for its **Key** attribute. This alias references a certificate in your Java KeyStore that will be used to check the signature validity.

Usually, Java KeyStores would contain a key pair (public and private keys) to be used for signing and validating messages for an specific server and the trusted public keys to be used to validate messages received from others servers.

Report a bug[88]

14.5.7. SAML2 Handlers

14.5.7.1. Introduction

When using PicketLink SAML Support, both IDP and SP need to be configured with *Handlers* . This handlers help the IDP and SP Authenticators to process SAML requests and responses.

[88] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29859-626907+%5BLatest%5D&comment=Title%3A+How+signatures+are+validated+%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29859-626907+02+Apr+2014+11%3A19+en-US+%5BLatest%5D

The handlers are basically an implementation of the Chain of Responsibility pattern (Gof). Each handler provides a specific logic about how to process SAML requests and responses.

Report a bug[89]

14.5.7.2. Configuring Handlers

The handlers are configures inside the picketlink.xml file. Here is how it looks like:

```
<Handlers xmlns="urn:picketlink:identity-federation:handler:config:2.1">
    <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler" />
    <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler" />
    <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler" />
    <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler" />
</Handlers>
```

Report a bug[90]

14.5.7.2.1. Handlers Element

This element defines a list of Handler elements.

Name	Description	Value
ChainClass	Defines the name of a class that implements the *org.picketlink. identity.federation. core.saml. v2.interfaces. SAML2HandlerChain* interface.	Defaults to **org.picketlink. identity.federation. core.saml. v2.impl. DefaultSAML2HandlerChain.**

Report a bug[91]

14.5.7.2.2. Handler Element

This element defines a specific Handler.

Name	Description
class	Defines the name of a class that implements the org.picketlink. identity.federation. core.saml. v2.interfaces. SAML2Handler interface.

[89] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29860-620466+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29860-620466+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[90] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29861-620467+%5BLatest%5D&comment=Title%3A+Configuring+Handlers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[91] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29862-620468+%5BLatest%5D&comment=Title%3A+Handlers+Element%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29862-620468+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

Report a bug[92]

14.5.7.3. Custom Handlers

PicketLink provide ways to you create your own handlers. Just create a class that implements the *org.picketlink.identity.federation.core.saml.v2.interfaces.SAML2Handler interface.*

Before create your own implementations, please take a look at the built-in handlers. They can help you a lot.

Report a bug[93]

14.5.7.4. Built-in Handlers

PicketLink as part of the SAMLv2 support has a number of handlers that need to be configured.

The Handlers are:

1. *Section 14.5.7.7, "SAML2AuthenticationHandler"*
2. *Section 14.5.7.6, "SAML2AttributeHandler"*
3. *Section 14.5.7.5, "RolesGenerationHandler"*
4. *Section 14.5.7.9, "SAML2IssuerTrustHandler"*
5. *Section 14.5.7.10, "SAML2LogOutHandler.java"*

Report a bug[94]

14.5.7.5. RolesGenerationHandler

14.5.7.5.1. Objective

Handler dealing with attributes for SAML2

Report a bug[95]

14.5.7.5.2. Fully Qualified Name

org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler

[92] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29863-620469+%5BLatest%5D&comment=Title%3A+Handler+Element%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29863-620469+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[93] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29864-620470+%5BLatest%5D&comment=Title%3A+Custom+Handlers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29864-620470+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[94] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29865-620694+%5BLatest%5D&comment=Title%3A+Built-in+Handlers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29865-620694+12+Mar+2014+12%3A36+en-US+%5BLatest%5D

[95] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29866-620472+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29866-620472+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

Report a bug[96]

14.5.7.5.3. Configuration

Should be configured in WEB-INF/picketlink-handlers.xml

Report a bug[97]

14.5.7.5.3.1. Example:

Example 14.6. WEB-INF/picketlink-handlers.xml

```
<Handlers xmlns="urn:picketlink:identity-federation:handler:config:1.0">
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler"/
>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler"/>
</Handlers>
```

Report a bug[98]

14.5.7.5.3.2. Configuration Parameters

#	Name	Type	Objective	Default Value	SP/IDP	Since Version
1	ATTRIBUTE_MANAGER	string	fqn of attribute manager class	org.picketlink.identity.federation.core.impl.EmptyAttributeManager	IDP	2.0

Report a bug[99]

[96] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29867-620473+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29867-620473+12+Mar+2014+12%3A32+en-US+%5BLatest%5D

[97] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29868-620474+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[98] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29869-620475+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29869-620475+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[99] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29870-620476+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29870-620476+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

14.5.7.5.3.3. Example:

Example 14.7. WEB-INF/picketlink-handlers.xml

```
<Handler class="org.picketlink.
                    identity.federation.
                    web.handlers.
                    saml2.RolesGenerationHandler">
<Option Key="ATTRIBUTE_MANAGER" Value="org.some.fun.class"/>
</Handler>
```

Report a bug[100]

14.5.7.6. SAML2AttributeHandler

14.5.7.6.1. Objective

Handler dealing with attributes for SAML2. On the SP side, it converts IDP-returned Attributes and stores them under the user's HttpSession. On the IDP side, converts the given HttpSession attributes into SAML Response Attributes. SP-side code can retrieve the Attributes from a Map stored under the session key GeneralConstants.SESSION_ATTRIBUTE_MAP.

Report a bug[101]

14.5.7.6.2. Fully Qualified Name

org.picketlink.identity.federation.web.handlers.saml2.SAML2AttributeHandler

Report a bug[102]

14.5.7.6.3. Configuration

Should be configured in WEB-INF/picketlink-handlers.xml

Report a bug[103]

[100] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29871-620477+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29871-620477+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[101] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29872-620478+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29872-620478+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[102] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29873-620479+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29873-620479+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[103] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29874-620480+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.5.7.6.3.1. Example:

Example 14.8. WEB-INF/picketlink-handlers.xml

```
<Handlers xmlns="urn:picketlink:identity-federation:handler:config:1.0">
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler"/
>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler"/>
</Handlers>
```

Report a bug[104]

14.5.7.6.4. Configuration Parameters

#	Name	Type	Objective	Default Value	SP/IDP	Since Version
1	ATTRIBUTE_MANAGER	string	fqn of attribute manager class	org.picketlink.identity.federation.core.impl.EmptyAttributeManager	IDP	2.0
2	ATTRIBUTE_KEYS	String	a comma separated list of string values representing attributes to be sent		IDP	2.0
3	ATTRIBUTE_CHOOSE_FRIENDLY_NAME	boolean	set to true if you require attributes to be keyed by friendly name rather than default name.		SP	2.0

Report a bug[105]

[104] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29875-620481+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29875-620481+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[105] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29876-620482+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.5.7.6.4.1. Example:

Example 14.9. WEB-INF/picketlink-handlers.xml

```
<Handler class="org.picketlink.
                    identity.federation.
                    web.handlers.
                    saml2.SAML2AttributeHandler">
<Option Key="ATTRIBUTE_CHOOSE_FRIENDLY_NAME" Value="true"/>
</Handler>
```

Report a bug[106]

14.5.7.6.4.2. Example:

Error

Topic 29879 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 29879* for more detailed information.

14.5.7.6.4.3. Additional References

- *PicketLink IDP using LDAP Attributes*[107]

Report a bug[108]

14.5.7.7. SAML2AuthenticationHandler

14.5.7.7.1. Objective

Handler handles the SAML request at the IDP and the SAML response at the SP.

Report a bug[109]

14.5.7.7.2. Fully Qualified Name

org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler

[106] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29877-620483+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29877-620483+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[107] https://community.jboss.org/wiki/PicketLinkIDPUsingLDAPAttributes

[108] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29882-626724+%5BLatest%5D&comment=Title%3A+Additional+References%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29882-626724+01+Apr+2014+16%3A24+en-US+%5BLatest%5D

[109] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29883-620489+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29883-620489+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

Report a bug[110]

14.5.7.7.3. Configuration

Should be configured in WEB-INF/picketlink-handlers.xml

Report a bug[111]

14.5.7.7.3.1. Example:

Example 14.10. WEB-INF/picketlink-handlers.xml

```
<Handlers xmlns="urn:picketlink:identity-federation:handler:config:1.0">
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler"/
>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler"/>
</Handlers>
```

Report a bug[112]

14.5.7.7.4. Configuration Parameters

#	Name	Type	Objective	SP/IDP	Since Version
1	CLOCK_ SKEW_ MILIS	string	a long value in miliseconds to add a clock skew to assertion expiration validation at the Service provider	SP	2.0
2	DISABLE_ AUTHN_ STATEMENT	boolean	Setting a value will disable the generation	IDP	2.0

[110] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29884-620490+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29884-620490+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[111] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29885-620491+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[112] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29886-620492+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29886-620492+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

#	Name	Type	Objective	SP/IDP	Since Version
			of an AuthnStatement		
3	DISABLE_ SENDING_ ROLES	boolean	Setting any value will disable the generation and return of roles to SP	IDP	2.0
4	DISABLE_ ROLE_ PICKING	boolean	Setting to true will disable picking IDP attribute statements	SP	2.0
5	ROLE_ KEY	String	a csv list of strings that represent the roles coming from IDP	SP	2.0
6	ASSERTION_ CONSUMER_ URL	String	the url to be used for assertionConsumerURL	SP	2.0
7	NAMEID_ FORMAT	String	Setting to a value will provide the nameid format to be sent to IDP	SP	2.0
8	ASSERTION_ SESSION_ ATTRIBUTE_ NAME	String	Specifies the name of the session attribute where the assertion will be stored. The assertion is stored as a DOM Document. This option is useful when you need to obtain the user's assertion to propagate or validate it against the STS.	SP	2.1.7
9	AUTHN_CONTEXT_CLASSES	String	Specifies a single or a comma separated list of SAML	SP	2.5.0

#	Name	Type	Objective	SP/IDP	Since Version
			Authentication Classes to be used when creating an AuthnRequest. The value can be a full qualified name (FQN) or an alias. For each standard class name there is an alias, as defined by the **`org.picketlink.common.constants.SAMLAuthentic`		
9	REQUESTED_AUTHN_CONTEXT_COMPARISON	String	Specifies the Comparison attribute of the RequestedAuthnContext. This option should be used in conjunction with the AUTHN_CONTEXT_CLASSES option. Only the values defined by the specification are supported.	SP	2.5.0

Report a bug[113]

14.5.7.7.4.1. Example:

Example 14.11. WEB-INF/picketlink-handlers.xml

```
<Handler class="org.picketlink.identity.
                    federation.web.
                    handlers.saml2.SAML2AuthenticationHandler">
<Option Key="DISABLE_ROLE_PICKING" Value="true"/>
</Handler>
```

Report a bug[114]

[113] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29887-620493+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[114] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29888-620494+%5BLatest

14.5.7.7.4.2. NAMEID_FORMAT:

The **transient** and **persistent nameid-formats** are used to obfuscate the actual identity in order to make linking activities extremely difficult between different SPs being served by the same IDP. A transient policy only lasts for the duration of the login session, where a persistent policy will reuse the obfuscated identity across multiple login sessions.

The Value can either be one of the following "official" values or a vendor-specific value supported by the IDP. Any string value is passed through to the NameIDPolicy's Format attribute as-is in an AuthnRequest.

urn:oasis:names:tc:SAML:2.0:nameid-format: **transient** urn:oasis:names:tc:SAML:2.0:nameid-format: **persistent** urn:oasis:names:tc:SAML:1.1:nameid-format: **unspecified** urn:oasis:names:tc:SAML:1.1:nameid-format: **emailAddress** urn:oasis:names:tc:SAML:1.1:nameid-format: **X509SubjectName** urn:oasis:names:tc:SAML:1.1:nameid-format: **WindowsDomainQualifiedName** urn:oasis:names:tc:SAML:2.0:nameid-format: **kerberos** urn:oasis:names:tc:SAML:2.0:nameid-format: **entity**

Report a bug[115]

14.5.7.8. SAML2EncryptionHandler

14.5.7.8.1. Objective

Handles SAML Assertions Encryption and Signature Generation. This handler uses the configuration provided in the *Section 14.5.6, "Digital Signatures in SAML Assertions"* to encrypt and sign SAML Assertions.

Report a bug[116]

14.5.7.8.2. Fully Qualified Name

org.picketlink.identity.federation.web.handlers.saml2.SAML2EncryptionHandler

Report a bug[117]

14.5.7.8.3. Restrictions

- This handler should be used only when configuring Identity Providers.
- For Service Providers, the decryption of SAML Assertion is already done by the authenticators.

%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29888-620494+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[115] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29889-620495+%5BLatest%5D&comment=Title%3A+NAMEID_FORMAT%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29889-620495+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[116] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29890-620695+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29890-620695+12+Mar+2014+12%3A37+en-US+%5BLatest%5D

[117] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29891-620497+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29891-620497+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

- When using this handler, make sure that your service providers are also configured with the *Section 14.5.7.11, "SAML2SignatureGenerationHandler"* and the *Section 14.5.7.12, "SAML2SignatureValidationHandler"* handlers.
- *Do not use this handler with the __ Section 14.5.7.11, "SAML2SignatureGenerationHandler" _* configured in the same chain. Otherwise SAML messages will be signed several times._

Report a bug[118]

14.5.7.8.4. Configuration

Should be configured in WEB-INF/picketlink.xml:

Report a bug[119]

14.5.7.8.4.1. Example:

Error

Topic 29894 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 29894* for more detailed information.

14.5.7.8.4.2. Configuration Parameters

#	Name	Type	Objective	Default Value	SP/IDP	Since Version

Report a bug[120]

14.5.7.9. SAML2IssuerTrustHandler

14.5.7.9.1. Objective

Handles Issuer trust.Trust decisions are based on the url of the issuer of the saml request/response sent to the handler chain.

[118] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29892-620696+%5BLatest%5D&comment=Title%3A+Restrictions%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29892-620696+12+Mar+2014+12%3A37+en-US+%5BLatest%5D

[119] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29893-620499+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[120] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29896-620502+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29896-620502+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

Report a bug[121]

14.5.7.9.2. Fully Qualified Name

org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler

Report a bug[122]

14.5.7.9.3. Configuration

Should be configured in WEB-INF/picketlink-handlers.xml

Report a bug[123]

14.5.7.9.3.1. Example:

Example 14.12. WEB-INF/picketlink-handlers.xml

```
<Handlers xmlns="urn:picketlink:identity-federation:handler:config:1.0">
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler"/
>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler"/>
</Handlers>
```

Report a bug[124]

14.5.7.9.3.2. Configuration Parameters

#	Name	Type	Objective	Default Value	SP/IDP	Since Version

[121] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29898-620504+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29898-620504+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[122] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29899-620505+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29899-620505+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[123] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29900-620506+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[124] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29901-620507+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29901-620507+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

Report a bug[125]

14.5.7.10. SAML2LogOutHandler.java

14.5.7.10.1. Objective

Handler for SAML2 Logout Profile.

Report a bug[126]

14.5.7.10.2. Fully Qualified Name

org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler.java

Report a bug[127]

14.5.7.10.3. Configuration

Should be configured in WEB-INF/picketlink-handlers.xml

Report a bug[128]

14.5.7.10.3.1. Example:

Example 14.13. WEB-INF/picketlink-handlers.xml

```
<Handlers xmlns="urn:picketlink:identity-federation:handler:config:1.0">
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler"/>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler"/
>
  <Handler
 class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler"/>
</Handlers>
```

[125] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29902-620508+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29902-620508+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[126] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29904-620510+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29904-620510+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[127] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29905-620511+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29905-620511+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[128] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29906-620512+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

Report a bug[129]

14.5.7.10.3.2. Configuration Parameters

#	Name	Type	Objective	Default Value	SP/IDP	Since Version

Report a bug[130]

14.5.7.11. SAML2SignatureGenerationHandler

14.5.7.11.1. Objective

Handles SAML Signature Generation. This handler uses the configuration provided in the *Section 14.5.6, "Digital Signatures in SAML Assertions"* to sign SAML messages.

Report a bug[131]

14.5.7.11.2. Fully Qualified Name

org.picketlink.identity.federation.web.handlers.saml2.SAML2SignatureGenerationHandler

Report a bug[132]

14.5.7.11.3. Configuration

Should be configured in WEB-INF/picketlink.xml.

Report a bug[133]

[129] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29907-620513+%5BLatest%5D&comment=Title%3A+Example%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29907-620513+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[130] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29908-620514+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29908-620514+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[131] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29910-620697+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29910-620697+12+Mar+2014+12%3A37+en-US+%5BLatest%5D

[132] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29911-620517+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29911-620517+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[133] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29912-620518+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.5.7.11.3.1. Example:

Error

Topic 29913 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 29913* for more detailed information.

14.5.7.11.3.2. Configuration Parameters

#	Name	Type	Objective	Default Value	SP/IDP	Since Version

Report a bug[134]

14.5.7.12. SAML2SignatureValidationHandler

14.5.7.12.1. Objective

Handles SAML Signature Validation. This handler uses the configuration provided in the *Section 14.5.6, "Digital Signatures in SAML Assertions"* to process signature validation.

Report a bug[135]

14.5.7.12.2. Fully Qualified Name

org.picketlink.identity.federation.web.handlers.saml2.SAML2SignatureValidationHandler

Report a bug[136]

14.5.7.12.3. Configuration

Should be configured in WEB-INF/picketlink.xml.

Report a bug[137]

[134] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29915-620521+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29915-620521+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[135] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29917-620698+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29917-620698+12+Mar+2014+12%3A37+en-US+%5BLatest%5D

[136] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29918-620524+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29918-620524+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[137] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29919-620525+%5BLatest

14.5.7.12.3.1. Example:

Error

Topic 29920 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 29920* for more detailed information.

14.5.7.12.3.2. Configuration Parameters

#	Name	Type	Objective	Default Value	SP/IDP	Since Version

Report a bug[138]

14.5.8. Single Logout

Table of Contents

Even though the SAML v2.0 specification has support for Global Logout, you have to use it very very wisely. Just remember that you need to keep the participants to a low number (say upto 5 participants with one IDP).

Global Logout : The user initiates GLO at one service provider which will log out the user at the IDP and all the service providers.

Local Logout : The user logs out of one service provider only. The session at the IDP and other service providers is intact.

Report a bug[139]

14.5.8.1. Configuring the GLO

The service provider url should be appended with "?GLO=true"

Basically, in the service provider page, have a url that has the query parameter.

Assume, your service provider is *http://localhost:8080/sales/,*[140] then the url for the global log out would be *http://localhost:8080/sales/?GLO=true*

%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[138] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29922-620528+%5BLatest%5D&comment=Title%3A+Configuration+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29922-620528+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[139] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29924-620530+%5BLatest%5D&comment=Title%3A+Single+Logout%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[140] http://localhost:8080/sales/,

Report a bug[141]

14.5.8.2. Configuring the LLO

The service provider url should be appended with "?LLO=true"

Basically, in the service provider page, have a url that has the query parameter.

Assume, your service provider is *http://localhost:8080/sales/,*[142] then the url for the local log out would be *http://localhost:8080/sales/?LLO=true*

When using LLO, you must be aware of some security implications. The user is only disconnect from the service provider from which he logged out, which means that the user's session in the identity provider and others service providers are still active. In other words, the user's SSO session is still active and he is still able to log in in any other service provider. We strongly recommend to always use the Single Logout Profile (GLO).

Important

In the case of LLO, the service provider invalidates the session and forwards to a default logout page (logout.jsp) .Custom logout page can be configured in picketlink.xml page. Please refer to Service Provider Configuration.

Report a bug[143]

14.5.9. SAML2 Configuration Providers

Table of Contents

It is possible to use different Configuration Providers at the IDP and SP.

The configuration providers will then be the sole configuration leaders (instead of picketlink.xml)

Report a bug[144]

[141] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29925-626767+%5BLatest%5D&comment=Title%3A+Configuring+the+GLO%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29925-626767+01+Apr+2014+20%3A38+en-US+%5BLatest%5D

[142] http://localhost:8080/sales/,

[143] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29926-626768+%5BLatest%5D&comment=Title%3A+Configuring+the+LLO%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29926-626768+01+Apr+2014+20%3A41+en-US+%5BLatest%5D

[144] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29927-620533+%5BLatest%5D&comment=Title%3A+SAML2+Configuration+Providers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.5.9.1. Configuration providers at the IDP

14.5.9.1.1. IDPMetadataConfigurationProvider

Fully Qualified Name: org.picketlink.identity.federation.web.config.IDPMetadataConfigurationProvider

How does it work?

You will need to provide the metadata file inside idp-metadata.xml and put it in the IDP web application classpath. Put it in WEB-INF/classes directory.

Report a bug[145]

14.5.9.2. Configuration Providers at the SP

14.5.9.2.1. SPPostMetadataConfigurationProvider

Fully Qualified Name:
org.picketlink.identity.federation.web.config.SPPostMetadataConfigurationProvider

Binding Supported: SAML2/HTTP-POST

Report a bug[146]

14.5.9.2.1.1. How does it work?

You will need to provide the metadata file inside sp-metadata.xml and put it in the SP web application classpath. Put it in WEB-INF/classes directory.

Remember, in the case of SP, the metadata file should have a IDPSSODescriptor as well as a SPSSODescriptor.

Report a bug[147]

14.5.9.2.2. SPRedirectMetadataConfigurationProvider

Fully Qualified Name:
org.picketlink.identity.federation.web.config.SPRedirectMetadataConfigurationProvider

Binding Supported: SAML2/HTTP-Redirect

[145] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29928-620534+%5BLatest%5D&comment=Title%3A+IDPMetadataConfigurationProvider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29928-620534+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[146] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29929-620535+%5BLatest%5D&comment=Title%3A+SPPostMetadataConfigurationProvider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[147] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29930-620536+%5BLatest%5D&comment=Title%3A+How+does+it+work%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29930-620536+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

Report a bug[148]

14.5.9.2.2.1. How does it work?

You will need to provide the metadata file inside sp-metadata.xml and put it in the SP web application classpath. Put it in WEB-INF/classes directory.

Remember, in the case of SP, the metadata file should have a IDPSSODescriptor as well as a SPSSODescriptor.

Report a bug[149]

14.5.9.2.3. What about Key Information and other configuration that comes via picketlink-idfed.xml?

Both the IDP and SP applications when provided with the saml configuration provider will be given a parsed representation of the WEB-INF/picketlink-idfed.xml, which implies that the IDPType and SPType coming out finally will be a merger of the configuration provider and the settings from picketlink-idfed.xml

Report a bug[150]

14.5.10. Metadata Support

Table of Contents

Report a bug[151]

14.5.10.1. Introduction

It is possible to use different Configuration Providers at the IDP and SP. The configuration providers will then be the sole configuration leaders (instead of picketlink.xml) or provide additional configuration.

PicketLink SAML Metadata Support is provided and configured by the following configuration providers implementations:

[148] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29931-620537+%5BLatest%5D&comment=Title%3A+SPRedirectMetadataConfigurationProvider%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[149] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29932-620538+%5BLatest%5D&comment=Title%3A+How+does+it+work%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29932-620538+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[150] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29933-620539+%5BLatest%5D&comment=Title%3A+What+about+Key+Information+and+other+configuration+that+comes+via+picketlink-idfed.xml%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29933-620539+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[151] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29934-620540+%5BLatest%5D&comment=Title%3A+Metadata+Support%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

Name	Description	Provider Type
org.picketlink. identity.federation. web.config. IDPMetadataConfigurationProvider	For Identity Providers	IDP
org.picketlink. identity.federation. web.config. SPPostMetadataConfigurationProvider	For Service Providers using HTTP-POST Binding	SP
org.picketlink.identity. federation.web. config.SPRedirectMetadataConfigurationProvider	For Service Providers using HTTP-REDIRECT Binding	SP

These providers allows you to define some additional configuration to your IDP or SP using a SAML Metadata XML Schema instance, merging them with the ones provided in your *WEB-INF/picketlink.xml* .

Report a bug[152]

14.5.10.2. Configuration

To configure the SAML Metadata Configuration Providers you need to follow these steps:

- Define the PicketLink Authenticator (SP or IDP valves) and provide the configuration provider class name as an attribute
- Depending if you're configuring an IDP or SP, provide a metadata file and put it on the classpath:
- For Identity Providers : WEB-INF/classes/idp-metadata.xml
- For Service Providers : WEB-INF/classes/sp-metadata.xml

Report a bug[153]

14.5.10.2.1. Configuring the PicketLink Authenticator

To configure one of the provided SAML Metadata configuration providers you just need to configure the PicketLink Authenticator with the **configProvider** parameter/attribute.

For Identity Providers you should have a configuration as follow:

```
<jboss-web>
  <security-domain>idp</security-domain>
  <context-root>idp-metadata</context-root>
  <valve>
     <class-
name>org.picketlink.identity.federation.bindings.tomcat.idp.IDPWebBrowserSSOValve</class-
name>
     <param>
```

[152] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29935-620541+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29935-620541+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[153] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29936-620542+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
      <param-name>configProvider</param-name>
      <param-
value>org.picketlink.identity.federation.web.config.IDPMetadataConfigurationProvider</param-
value>
     </param>
   </valve>
</jboss-web>
```

For Service Providers you should have a configuration as follow:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
  <security-domain>sp</security-domain>
  <context-root>sales-metadata</context-root>
    <valve>
      <class-
name>org.picketlink.identity.federation.bindings.tomcat.sp.ServiceProviderAuthenticator</
class-name>
      <param>
 <param-name>configProvider</param-name>
 <param-
value>org.picketlink.identity.federation.web.config.SPPostMetadataConfigurationProvider</
param-value>
      </param>
    </valve>
</jboss-web>
```

Report a bug[154]

14.5.10.2.2. What about Key Information and other configuration that comes via picketlink-idfed.xml?

Both the IDP and SP applications when provided with the saml configuration provider will be given a parsed representation of the WEB-INF/picketlink.xml, which implies that the IDPType and SPType coming out finally will be a merger of the configuration provider and the settings from picketlink.xml

Report a bug[155]

14.5.10.3. Examples

The *PicketLink Quickstarts*[156] provide some examples for the SAML Metadata Support. Please check the following provided quickstarts:

- *https://github.com/picketlink/picketlink-quickstarts/tree/master/saml/idp-metadata*
- *https://github.com/picketlink/picketlink-quickstarts/tree/master/saml/sales-metadata*

[154] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29937-620543+%5BLatest%5D&comment=Title%3A+Configuring+the+PicketLink+Authenticator%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29937-620543+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[155] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29938-620544+%5BLatest%5D&comment=Title%3A+What+about+Key+Information+and+other+configuration+that+comes+via+picketlink-idfed.xml%3F%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29938-620544+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[156] https://docs.jboss.org/author/pages/viewpage.action?pageId=23986289

Report a bug[157]

14.5.11. Token Registry

14.5.11.1. Introduction

PicketLink supports the concept of *Token Registry* to store tokens using any store such databases, filesystem or memory.

They are useful for auditing and to track the tokens that were issued or revocated by the Identity Provider or the Security Token Service.

Tip

When running PicketLink in a clustered environment, consider using Token Registries with databases. That way changes to the token table are visible to all nodes.

Report a bug[158]

14.5.11.2. of-box Token Registries

The table bellow shows all implementations provided by PicketLink:

Name	Description	Version
org.picketlink. identity.federation. core.sts. registry.DefaultTokenRegistry	In-memory based registry. *Used by default if no configuration is provided*	2.x.x
org.picketlink. identity.federation. core.sts. registry.FileBasedTokenRegistry	Filesystem based registry	2.x.x
org.picketlink. identity.federation. core.sts. registry.JPABasedTokenRegistry	Database/JPA based registry	2.1.3

Report a bug[159]

[157] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29939-626769+%5BLatest%5D&comment=Title%3A+Examples%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29939-626769+01+Apr+2014+20%3A42+en-US+%5BLatest%5D

[158] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29940-620546+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29940-620546+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[159] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29941-620547+%5BLatest%5D&comment=Title%3A+of-box+Token+Registries%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29941-620547+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

14.5.11.3. Configuration

Token Registries are configured through the **PicketLinkSTS** (Security Token Service configuration) element in the **WEB-INF/picketlink.xml** file:

Tip

Read the documentation for more information about the *Section 14.5.3.6, "Security Token Service Configuration"* element and the *Section 14.5.3.6, "Security Token Service Configuration"* .

```
<PicketLinkSTS xmlns="urn:picketlink:identity-federation:config:1.0" TokenTimeout="5000"
 ClockSkew="0">
        <TokenProviders>
  <TokenProvider

 ProviderClass="org.picketlink.identity.federation.core.saml.v2.providers.SAML20AssertionTokenProvider"
   TokenType="urn:oasis:names:tc:SAML:2.0:assertion"
   TokenElement="Assertion" TokenElementNS="urn:oasis:names:tc:SAML:2.0:assertion">
   <Property Key="TokenRegistry"
 Value="org.picketlink.identity.federation.core.sts.registry.JPABasedTokenRegistry" />
  </TokenProvider>
 </TokenProviders>
</PicketLinkSTS>
```

The example above uses a SAML v2 Token Provider configured with the *org.picketlink.identity.federation.core.sts.registry.JPABasedTokenRegistry* implementation. This is done by the **TokenRegistry** property.

Report a bug[160]

14.5.11.3.1. org.picketlink.identity.federation.core.sts.registry.FileBasedTokenRegistry

```
<TokenProvider
 ProviderClass="org.picketlink.identity.federation.core.saml.v2.providers.SAML20AssertionTokenProvider"
 TokenType="urn:oasis:names:tc:SAML:2.0:assertion"
 TokenElement="Assertion" TokenElementNS="urn:oasis:names:tc:SAML:2.0:assertion">
    <Property Key="TokenRegistry" Value="FILE" />
    <Property Key="TokenRegistryFile" Value="/some/dir/token.registry" />
</TokenProvider>
```

Use the **TokenRegistryFile** to specify a file where the tokens should be persisted.

Report a bug[161]

[160] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29942-620699+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[161] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29943-620549+%5BLatest%5D&comment=Title%3A+org.picketlink.identity.federation.core.sts.registry.FileBasedTokenRegistry%0A

14.5.11.3.2. org.picketlink.identity.federation.core.sts.registry.JPABasedTokenRegistry

```
<TokenProvider
 ProviderClass="org.picketlink.identity.federation.core.saml.v2.providers.SAML20AssertionTokenProvider"
 TokenType="urn:oasis:names:tc:SAML:2.0:assertion"
 TokenElement="Assertion" TokenElementNS="urn:oasis:names:tc:SAML:2.0:assertion">
     <Property Key="TokenRegistry"
 Value="org.picketlink.identity.federation.core.sts.registry.JPABasedTokenRegistry" />
</TokenProvider>
```

This implementation requires that you have a valid JPA Persistence Unit named **picketlink-sts.**

Report a bug[162]

14.5.11.4. Custom Token Registry

If none of the built-in implementations are useful for you, PicketLink allows you to create your own implementation. To do that, just create a class that implements the **org.picketlink.identity.federation.core.sts.registry.SecurityTokenRegistry** interface.

Tip

We recommend that you take a look first at one of the provided implementation before building your own.

Bellow is an skeleton for a custom Token Registry implementation:

```
public class CustomSecurityTokenRegistry implements SecurityTokenRegistry {

    @Override
    public void addToken(String tokenID, Object token) throws IOException {
        // TODO: logic to add a token to the registry

    }

    @Override
    public void removeToken(String tokenID) throws IOException {
        // TODO: logic to remove a token to the registry
    }

    @Override
    public Object getToken(String tokenID) {
        // TODO: logic to get a token from the registry
        return null;
    }

}
```

%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29943-620549+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[162] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29944-620550+%5BLatest%5D&comment=Title%3A+org.picketlink.identity.federation.core.sts.registry.JPABasedTokenRegistry%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29944-620550+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

Report a bug[163]

14.5.12. Standalone vs JBossAS Distribution

PicketLink has SAMLv2 support for both JBossAS and a regular servlet container. The JBoss AS version contains deeper integration with the web container security such that you can make use of api such as request.getUserPrincipal() etc. Plus you can configure your favorite JAAS login module for authentication at the IDP side.

So, choose the *JBossAS version of PicketLink*[164] . If you do not run on JBoss AS or Apache Tomcat, then choose the *Section 14.5.13, "Standalone Web Applications(All Servlet Containers)"* .

Report a bug[165]

14.5.13. Standalone Web Applications(All Servlet Containers)

If your IDP or SP applications are not running on JBoss Application Server or Apache Tomcat, then you can use the standalone mode of PicketLink.

Report a bug[166]

14.5.13.1. Service Provider Configuration

In your web.xml, configure a *Section 14.5.13.6, "SPFilter"* as shown below as an example:

Example 14.14. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_2_5.xsd"
   version="2.5">

   <description>Sales Standalone Application</description>

   <filter>
     <description>
        The SP Filter intersects all requests at the SP and sees if there is a need to
 contact the IDP.
     </description>
     <filter-name>SPFilter</filter-name>
     <filter-class>org.picketlink.identity.federation.web.filters.SPFilter</filter-class>
     <init-param>
```

[163] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29945-620551+%5BLatest%5D&comment=Title%3A+Custom+Token+Registry%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29945-620551+12+Mar+2014+12%3A33+en-US+%5BLatest%5D

[164] https://docs.jboss.org/author/display/PLINK/Tomcat+Authenticators+%28Tomcat%2CJBossAS%29

[165] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29946-626770+%5BLatest%5D&comment=Title%3A+Standalone+vs+JBossAS+Distribution%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29946-626770+01+Apr+2014+20%3A42+en-US+%5BLatest%5D

[166] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29947-620553+%5BLatest%5D&comment=Title%3A+Standalone+Web+Applications%28All+Servlet+Containers%29%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
      <param-name>ROLES</param-name>
      <param-value>sales,manager</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>SPFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

After the SAML workflow is completed, the user principal is available in the http session at "picketlink.principal".

Something like,

import org.picketlink.identity.federation.web.constants.GeneralConstants;

Principal userPrincipal = (Principal) session.getAttribute(GeneralConstants.PRINCIPAL_ID);

Report a bug[167]

14.5.13.2. IDP Configuration

For an IDP web application to be SAML enabled on any Servlet Container, you will have to add listeners and servlets as shown in the web.xml below:

Part of the **idp-standalone.war**

Example 14.15. web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_2_5.xsd"
   version="2.5">

  <display-name>Standalone IDP</display-name>
  <description>
    IDP Standalone Application
  </description>

  <!-- Listeners -->
  <listener>
    <listener-class>org.picketlink.identity.federation.web.core.IdentityServer</listener-
class>
  </listener>


  <!-- Create the servlet -->
  <servlet>
    <servlet-name>IDPLoginServlet</servlet-name>
```

[167] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29948-620701+%5BLatest%5D&comment=Title%3A+Service+Provider+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29948-620701+12+Mar+2014+12%3A37+en-US+%5BLatest%5D

```
    <servlet-class>org.picketlink.identity.federation.web.servlets.IDPLoginServlet</
servlet-class>
  </servlet>
  <servlet>
    <servlet-name>IDPServlet</servlet-name>
    <servlet-class>org.picketlink.identity.federation.web.servlets.IDPServlet</servlet-
class>
  </servlet>

  <servlet-mapping>
    <servlet-name>IDPLoginServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>IDPServlet</servlet-name>
    <url-pattern>/IDPServlet</url-pattern>
  </servlet-mapping>

</web-app>
```

A jsp for login would be:

Example 14.16. jsp/login.jsp

```
<html><head><title>Login Page</title></head>
<body>
<font size='5' color='blue'>Please Login</font><hr>

<form action='<%=application.getContextPath()%>/' method='post'>
<table>
 <tr><td>Name:</td>
   <td><input type='text' name='JBID_USERNAME'></td></tr>
 <tr><td>Password:</td>
   <td><input type='password' name='JBID_PASSWORD' size='8'></td>
 </tr>
</table>
<br>
  <input type='submit' value='login'>
</form></body>
 </html>
```

The jsp for error would be:

Example 14.17. jsp/error.jsp

```
<html> <head> <title>Error!</title></head>
<body>

<font size='4' color='red'>
  The username and password you supplied are not valid.
</p>
Click <a href='<%= response.encodeURL("login.jsp") %>'>here</a>
to retry login

</body>
</form>
</html>
```

Report a bug[168]

14.5.13.3. Other References

1. *http://community.jboss.org/wiki/PicketLinkSAMLSSOForWebContainers*

Report a bug[169]

14.5.13.4. IDPLoginServlet

IDPLoginServlet provides the login capabilities for IDP applications running on any servlet container.

Report a bug[170]

14.5.13.4.1. Initialization Parameters

#	Name	Type	Objective	Default	Since
1	loginClass	String	fqn of an implementation of the ILoginHandler interface. Provides the authentication/ authorization.	org.picketlink. identity.federation. web.handlers. DefaultLoginHandler	2.0

Report a bug[171]

14.5.13.4.2. Configuration

The IDP application needs to contain **/jsp/login.jsp** . The jsp file needs to have a form with two text fields namely: JBID_USERNAME and JBID_PASSWORD to indicate username and password.

[168] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29949-620555+%5BLatest%5D&comment=Title%3A+IDP+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29949-620555+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[169] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29950-626771+%5BLatest%5D&comment=Title%3A+Other+References%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29950-626771+01+Apr+2014+20%3A43+en-US+%5BLatest%5D

[170] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29951-620557+%5BLatest%5D&comment=Title%3A+IDPLoginServlet%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[171] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29952-620558+%5BLatest%5D&comment=Title%3A+Initialization+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29952-620558+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

On successful authentication, this servlet redirects to the IDPServlet.

Report a bug[172]

14.5.13.5. IDPServlet

IDPServlet supports the SAMLv2 HTTP/POST binding for an IDP running on any servlet container.

Report a bug[173]

14.5.13.5.1. Initialization Parameters

#	Name	Type	Objective	Default	Since
1	CONFIG_ PROVIDER	String	optional - fqn of an implementation of the SAMLConfigurationProvider interface.		2.0
2	SIGN_ OUTGOING_ MESSAGES	boolean	optional - whether the IDP should sign outgoing messages	true	2.0
3	ROLE_ GENERATOR	String	optional - fqn of a RoleGenerator	org.picketlink. identity.federation. web.roles. DefaultRoleGenerator	2.0
4	ATTRIBUTE_ KEYS	String	optional - comma separated list of keys for attributes that need to be sent		2.0
5	IDENTITY_ PARTICIPANT_ STACK	String	optional - fqn of a custom IdentityParticipantStack implementation		2.0

Report a bug[174]

[172] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29953-620559+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29953-620559+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[173] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29954-620560+%5BLatest%5D&comment=Title%3A+IDPServlet%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[174] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29955-620561+%5BLatest

14.5.13.5.2. Configuration

The *Section 14.5.13.4, "IDPLoginServlet"* that is configured in the web application authenticates the user. The IDPServlet then sends back the SAML response message with the SAML assertion back to the Service Provider(SP).

Report a bug[175]

14.5.13.6. SPFilter

SPFilter is the filter that service provider applications need to have to provide HTTP/POST binding of the SAMLv2 specification for web applications running on any servlet container.

Report a bug[176]

14.5.13.6.1. Initialization Parameters

#	Name	Type	Objective	Default	Since
1	IGNORE_ SIGNATURES	boolean	optional - should the SP ignore signatures	false	2.0
2	SAML_ HANDLER_ CHAIN_ CLASS	String	optional - fqn of custom SAMLHandlerChain interface		2.0
3	ROLE_ VALIDATOR	String	optional - fqn of a IRoleValidator interface	org.picketlink. identity.federation. web.roles. DefaultRoleValidator	2.0
4	ROLES	String	optional - comma separated list of roles that the sp will take		2.0
5	LOGOUT_ PAGE	String	optional - a logout page	/logout.jsp	2.0

%5D&comment=Title%3A+Initialization+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29955-620561+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[175] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29956-620702+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29956-620702+12+Mar+2014+12%3A37+en-US+%5BLatest%5D

[176] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29957-620563+%5BLatest%5D&comment=Title%3A+SPFilter%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

Report a bug[177]

14.6. SAML v1.1

14.6.1. SAML v1.1

Please refer to the wikipedia *page*[178] for more information.

Report a bug[179]

14.6.2. PicketLink SAML v1.1 Support

Please read it at *http://community.jboss.org/wiki/PicketLinkSAMLV11Support*

Report a bug[180]

14.7. Trust

14.7.1. Security Token Server (STS)

14.7.1.1. Introduction

The WS-Trust specification defines extensions that build on WS-Security to provide a framework for requesting and issuing security tokens. Particularly, WS-Trust defines the concept of a security token service (STS), a service that can issue, cancel, renew and validate security tokens, and specifies the format of security token request and response messages.

Tip

Please look at the *PicketLink Quickstarts*[181] for the PicketLink Identity Provider web application. The quickstarts are useful resources where you can get configuration files.

[177] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29958-620564+%5BLatest%5D&comment=Title%3A+Initialization+Parameters%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29958-620564+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[178] http://en.wikipedia.org/wiki/SAML_1.1

[179] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29959-626703+%5BLatest%5D&comment=Title%3A+SAML+v1.1%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29959-626703+01+Apr+2014+16%3A10+en-US+%5BLatest%5D

[180] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29960-626701+%5BLatest%5D&comment=Title%3A+PicketLink+SAML+v1.1+Support%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29960-626701+01+Apr+2014+16%3A09+en-US+%5BLatest%5D

[181] https://docs.jboss.org/author/pages/viewpage.action?pageId=23986289

Report a bug[182]

14.7.1.2. References

PicketLink STS Dashboard[183]

Report a bug[184]

14.7.1.3. PicketLink JBoss Web Services Handlers

Page to list all the JBoss Web Services handlers that are part of the PicketLink project.

1. SAML2Handler
2. BinaryTokenHandler
3. WSAuthenticationHandler
4. WSAuthorizationHandler

Report a bug[185]

14.7.1.3.1. BinaryTokenHandler

14.7.1.3.1.1. Fully Qualified Name

org.picketlink.trust.jbossws.handler.BinaryTokenHandler

Report a bug[186]

14.7.1.3.1.2. Objective

A JBoss Web Services Handler that is stack agnostic that can be added on the client side to either pick a http header or cookie, that contains a binary token.

Report a bug[187]

---

[182] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29961-626700+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29961-626700+01+Apr+2014+16%3A08+en-US+%5BLatest%5D

[183] http://community.jboss.org/wiki/PicketLinkSTSDashboard

[184] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29962-626702+%5BLatest%5D&comment=Title%3A+References%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29962-626702+01+Apr+2014+16%3A09+en-US+%5BLatest%5D

[185] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29963-620569+%5BLatest%5D&comment=Title%3A+PicketLink+JBoss+Web+Services+Handlers%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[186] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29964-620570+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29964-620570+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[187] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29965-620571+%5BLatest

14.7.1.3.1.3. Author

Anil Saldhana

Report a bug[188]

14.7.1.3.1.4. Settings

Configuration:

System Properties:

- binary.http.header: http header name
- binary.http.cookie: http cookie name
- binary.http.encodingType: attribute value of the EncodingType attribute
- binary.http.valueType: attribute value of the ValueType attribute
- binary.http.valueType.namespace: namespace for the ValueType attribute
- binary.http.valueType.prefix: namespace for the ValueType attribute
- binary.http.cleanToken: true or false dependending on whether the binary token has to be cleaned

Setters:

Please see the see also section.See Also:setHttpHeaderName(String)setHttpCookieName(String)setEncodingType(String)setValueType(String)setValue

Report a bug[189]

14.7.1.3.1.5. Test Case

http://anonsvn.jboss.org/repos/picketlink/integration-tests/trunk/picketlink-trust-tests/src/test/java/org/picketlink/test/trust/tests/STSWSBinaryTokenTestCase.java

Report a bug[190]

14.7.1.3.2. SAML2Handler

14.7.1.3.2.1. Full Name:

org.picketlink.trust.jbossws.handler.SAML2Handler

%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29965-620571+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[188] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29966-620572+%5BLatest%5D&comment=Title%3A+Author%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29966-620572+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[189] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29967-620573+%5BLatest%5D&comment=Title%3A+Settings%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29967-620573+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[190] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29968-626699+%5BLatest%5D&comment=Title%3A+Test+Case%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29968-626699+01+Apr+2014+16%3A08+en-US+%5BLatest%5D

Report a bug[191]

14.7.1.3.2.2. Authors:

- Marcus Moyses
- Anil Saldhana

Report a bug[192]

14.7.1.3.2.3. Objective:

This is a JBossWS handler (stack agnostic) that supports the SAML token profile of the Oasis Web Services Security (WSS) standard.

It can be configured both on the client side and the server side. The configuration is shown below both the client(outbound) as well as server(inbound).

Report a bug[193]

14.7.1.3.2.3.1. Outbound:

This is the behavior when the handler is configured on the client side.

The client side usage is shown in the following client class. If you need to use an XML file to specify the handler on the client side, then please look in the references section below.

Example 14.18. STSWSClientTestCase.java

```
package org.picketlink.test.trust.tests;

import java.net.URL;
import java.util.List;

import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;
import javax.xml.ws.handler.Handler;

import org.junit.Test;
import org.picketlink.identity.federation.api.wstrust.WSTrustClient;
import org.picketlink.identity.federation.api.wstrust.WSTrustClient.SecurityInfo;
import org.picketlink.identity.federation.core.wstrust.WSTrustException;
import org.picketlink.identity.federation.core.wstrust.plugins.saml.SAMLUtil;
import org.picketlink.test.trust.ws.WSTest;
import org.picketlink.trust.jbossws.SAML2Constants;
import org.picketlink.trust.jbossws.handler.SAML2Handler;
```

[191] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29969-620575+%5BLatest%5D&comment=Title%3A+Full+Name%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29969-620575+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[192] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29970-620576+%5BLatest%5D&comment=Title%3A+Authors%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29970-620576+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[193] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29971-620577+%5BLatest%5D&comment=Title%3A+Objective%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
import org.w3c.dom.Element;

/**
 * A Simple WS Test for the SAML Profile of WSS
 * @author Marcus Moyses
 * @author Anil Saldhana
 */
public class STSWSClientTestCase
{
   private static String username = "UserA";
   private static String password = "PassA";

   @SuppressWarnings("rawtypes")
   @Test
   public void testWSInteraction() throws Exception {
      WSTrustClient client = new WSTrustClient("PicketLinkSTS", "PicketLinkSTSPort",
            "http://localhost:8080/picketlink-sts/PicketLinkSTS",
            new SecurityInfo(username, password));
    Element assertion = null;
    try {
        System.out.println("Invoking token service to get SAML assertion for " +
 username);
        assertion = client.issueToken(SAMLUtil.SAML2_TOKEN_TYPE);
        System.out.println("SAML assertion for " + username + " successfully obtained!");
    } catch (WSTrustException wse) {
        System.out.println("Unable to issue assertion: " + wse.getMessage());
        wse.printStackTrace();
        System.exit(1);
    }

    URL wsdl = new URL("http://localhost:8080/picketlink-wstest-tests/WSTestBean?wsdl");
    QName serviceName = new QName("http://ws.trust.test.picketlink.org/",
 "WSTestBeanService");
    Service service = Service.create(wsdl, serviceName);
    WSTest port = service.getPort(new QName("http://ws.trust.test.picketlink.org/",
 "WSTestBeanPort"), WSTest.class);
    BindingProvider bp = (BindingProvider)port;
    bp.getRequestContext().put(SAML2Constants.SAML2_ASSERTION_PROPERTY, assertion);
    List<Handler> handlers = bp.getBinding().getHandlerChain();
    handlers.add(new SAML2Handler());
    bp.getBinding().setHandlerChain(handlers);

    port.echo("Test");
   }
}
```

Note: the SAML2Handler is instantiated and added to the handler list that is obtained from the BindingProvider binding.

There are two ways by which the SAML2Handler picks the SAML2 Assertion to send via the SOAP message.

- The Client can push the SAML2 Assertion into the SOAP MessageContext under the key " **org.picketlink.trust.saml.assertion** ". In the example code above, look in the call bindingProvider.getRequestContext().put(xxxxx)
- The SAML2 Assertion is available as part of the JAAS subject on the security context. This can happen if there has been a JAAS interaction with the usage of PicketLink STS login modules.

Report a bug[194]

14.7.1.3.2.3.2. Inbound:

This is the behavior when the handler is configured on the server side.

The server side setting is as follows:

Example 14.19. handlers.xml

```
<?xml version="1.0" encoding="UTF-8"?>


<handler-chains xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ns1="http://org.jboss.ws/jaxws/samples/logicalhandler"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee javaee_web_services_1_2.xsd">



  <handler-chain>
    <handler>
      <handler-name>SAML2Handler</handler-name>
      <handler-class>org.picketlink.trust.jbossws.handler.SAML2Handler</handler-class>
    </handler>
  </handler-chain>


</handler-chains>
```

The SAML2Handler looks for a SAML2 Assertion on the SOAP message. If it is available then it constructs a SamlCredential object with the assertion and then sets it on the SecurityContext for the JAAS layer to authenticate the call.

Report a bug[195]

14.7.1.3.2.4. References

JBossWS User Guide on Handlers[196]

JBossWS JAXWS Client Configuration[197]

Report a bug[198]

---

[194] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29972-620578+%5BLatest%5D&comment=Title%3A+Outbound%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29972-620578+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[195] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29973-620579+%5BLatest%5D&comment=Title%3A+Inbound%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29973-620579+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[196] http://community.jboss.org/wiki/JBossWS-UserGuide#Handler_Framework

[197] http://community.jboss.org/wiki/JBossWS-JAX-WSClientConfiguration

[198] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29974-626698+%5BLatest%5D&comment=Title%3A+References%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29974-626698+01+Apr+2014+16%3A08+en-US+%5BLatest%5D

14.7.1.3.3. WSAuthenticationHandler

14.7.1.3.3.1. FQN:

org.picketlink.trust.jbossws.handler.WSAuthenticationHandler

Report a bug[199]

14.7.1.3.3.2. Objective:

Perform authentication for POJO based webservices.

Report a bug[200]

14.7.1.3.3.3. Example Usage:

Assume that you have a POJO.

```
package org.picketlink.test.trust.ws;

import javax.jws.HandlerChain;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

/**
 * POJO that is exposed as WS
 * @author Anil Saldhana
 */
@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
@HandlerChain(file="authorize-handlers.xml")
public class POJOBean
{
   @WebMethod
   public void echo(String echo)
   {
      System.out.println(echo);
   }

   @WebMethod
   public void echoUnchecked(String echo)
   {
      System.out.println(echo);
   }
}
```

Note the use of the @HandlerChain annotation that defines the handler xml.

The handler xml is authorize-handlers.xml.

[199] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29975-620581+%5BLatest%5D&comment=Title%3A+FQN%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29975-620581+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[200] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29976-620582+%5BLatest%5D&comment=Title%3A+Objective%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29976-620582+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

```
<?xml version="1.0" encoding="UTF-8"?>


<handler-chains xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee javaee_web_services_1_2.xsd">

  <handler-chain>


    <handler>
      <handler-name>WSAuthorizationHandler</handler-name>
      <handler-class>org.picketlink.trust.jbossws.handler.WSAuthorizationHandler</handler-
class>
    </handler>

    <handler>
      <handler-name>WSAuthenticationHandler</handler-name>
      <handler-class>org.picketlink.trust.jbossws.handler.WSAuthenticationHandler</handler-
class>
    </handler>

    <handler>
      <handler-name>SAML2Handler</handler-name>
      <handler-class>org.picketlink.trust.jbossws.handler.SAML2Handler</handler-class>
    </handler>


  </handler-chain>


</handler-chains>
```

Warning

Note : The order of execution of the handlers is SAML2Handler, WSAuthenticationHandler and WSAuthorizationHandler. These need to be defined in reverse order in the xml.

Since we intend to expose a POJO as a webservice, we need to package in a web archive (war).

The web.xml is:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"
 version="2.5">

 <servlet>
  <display-name>POJO Web Service</display-name>
  <servlet-name>POJOBeanService</servlet-name>
  <servlet-class>org.picketlink.test.trust.ws.POJOBean</servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>POJOBeanService</servlet-name>
  <url-pattern>/POJOBeanService</url-pattern>
 </servlet-mapping>
</web-app>
```

Warning

Please do not define any <security-constraint> in the web.xml

The jboss-web.xml is:

```
<jboss-web>
  <security-domain>sts</security-domain>
</jboss-web>
```

The jboss-wsse.xml is

```
<jboss-ws-security xmlns="http://www.jboss.com/ws-security/config"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://www.jboss.com/ws-security/config
                   http://www.jboss.com/ws-security/schema/jboss-ws-security_1_0.xsd">

  <port name="POJOBeanPort">
    <operation name="{http://ws.trust.test.picketlink.org/}echoUnchecked">
      <config>
        <authorize>
          <unchecked/>
        </authorize>
      </config>
    </operation>

    <operation name="{http://ws.trust.test.picketlink.org/}echo">
      <config>
        <authorize>
          <role>JBossAdmin</role>
        </authorize>
      </config>
    </operation>
  </port>



</jboss-ws-security>
```

As you can see, there are two operations defined on the POJO web services and each of these operations require different access control. The echoUnchecked() method allows free access to any authenticated user whereas the echo() method requires the caller to have "JBossAdmin" role.

The war should look as:

```
anil@localhost:~/picketlink/picketlink/integration-tests/trunk/picketlink-trust-tests$ jar
 tvf target/pojo-test.war
     0 Mon Apr 11 19:48:32 CDT 2011 META-INF/
   123 Mon Apr 11 19:48:30 CDT 2011 META-INF/MANIFEST.MF
     0 Mon Apr 11 19:48:30 CDT 2011 WEB-INF/
     0 Mon Apr 11 19:48:30 CDT 2011 WEB-INF/classes/
     0 Mon Apr 11 19:48:30 CDT 2011 WEB-INF/classes/org/
     0 Mon Apr 11 19:48:30 CDT 2011 WEB-INF/classes/org/picketlink/
     0 Mon Apr 11 19:48:30 CDT 2011 WEB-INF/classes/org/picketlink/test/
     0 Mon Apr 11 19:48:30 CDT 2011 WEB-INF/classes/org/picketlink/test/trust/
     0 Mon Apr 11 19:48:30 CDT 2011 WEB-INF/classes/org/picketlink/test/trust/ws/
     0 Mon Apr 11 19:48:30 CDT 2011 WEB-INF/lib/
   858 Mon Apr 11 19:48:26 CDT 2011 WEB-INF/classes/authorize-handlers.xml
```

```
  1021 Mon Apr 11 19:48:28 CDT 2011 WEB-INF/classes/org/picketlink/test/trust/ws/
POJOBean.class
    65 Mon Apr 11 12:00:32 CDT 2011 WEB-INF/jboss-web.xml
   770 Mon Apr 11 17:44:16 CDT 2011 WEB-INF/jboss-wsse.xml
   598 Mon Apr 11 16:25:46 CDT 2011 WEB-INF/web.xml
     0 Mon Apr 11 19:48:32 CDT 2011 META-INF/maven/
     0 Mon Apr 11 19:48:32 CDT 2011 META-INF/maven/org.picketlink/
     0 Mon Apr 11 19:48:32 CDT 2011 META-INF/maven/org.picketlink/picketlink-integration-
trust-tests/
  7918 Mon Apr 11 18:56:16 CDT 2011 META-INF/maven/org.picketlink/picketlink-integration-
trust-tests/pom.xml
   142 Mon Apr 11 19:48:30 CDT 2011 META-INF/maven/org.picketlink/picketlink-integration-
trust-tests/pom.properties
anil@localhost:~/picketlink/picketlink/integration-tests/trunk/picketlink-trust-tests
```

The Test Case is something like:

```
 package org.picketlink.test.trust.tests;

import java.net.URL;
import java.util.List;

import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;
import javax.xml.ws.handler.Handler;

import org.junit.Test;
import org.picketlink.identity.federation.api.wstrust.WSTrustClient;
import org.picketlink.identity.federation.api.wstrust.WSTrustClient.SecurityInfo;
import org.picketlink.identity.federation.core.wstrust.WSTrustException;
import org.picketlink.identity.federation.core.wstrust.plugins.saml.SAMLUtil;
import org.picketlink.test.trust.ws.WSTest;
import org.picketlink.trust.jbossws.SAML2Constants;
import org.picketlink.trust.jbossws.handler.SAML2Handler;
import org.w3c.dom.Element;

/**
 * A Simple WS Test for POJO WS Authorization using PicketLink
 * @author Anil Saldhana
 * @since Oct 3, 2010
 */
public class POJOWSAuthorizationTestCase
{
   private static String username = "UserA";
   private static String password = "PassA";

   @SuppressWarnings("rawtypes")
   @Test
   public void testWSInteraction() throws Exception
   {
      // Step 1:  Get a SAML2 Assertion Token from the STS
      WSTrustClient client = new WSTrustClient("PicketLinkSTS", "PicketLinkSTSPort",
            "http://localhost:8080/picketlink-sts/PicketLinkSTS",
            new SecurityInfo(username, password));
      Element assertion = null;
      try {
         System.out.println("Invoking token service to get SAML assertion for " + username);
         assertion = client.issueToken(SAMLUtil.SAML2_TOKEN_TYPE);
         System.out.println("SAML assertion for " + username + " successfully obtained!");
      } catch (WSTrustException wse) {
         System.out.println("Unable to issue assertion: " + wse.getMessage());
         wse.printStackTrace();
         System.exit(1);
      }
```

```
      // Step 2: Stuff the Assertion on the SOAP message context and add the SAML2Handler to
 client side handlers
      URL wsdl = new URL("http://localhost:8080/pojo-test/POJOBeanService?wsdl");
      QName serviceName = new QName("http://ws.trust.test.picketlink.org/",
 "POJOBeanService");
      Service service = Service.create(wsdl, serviceName);
      WSTest port = service.getPort(new QName("http://ws.trust.test.picketlink.org/",
 "POJOBeanPort"), WSTest.class);
      BindingProvider bp = (BindingProvider)port;
      bp.getRequestContext().put(SAML2Constants.SAML2_ASSERTION_PROPERTY, assertion);
      List<Handler> handlers = bp.getBinding().getHandlerChain();
      handlers.add(new SAML2Handler());
      bp.getBinding().setHandlerChain(handlers);

      //Step 3: Access the WS. Exceptions will be thrown anyway.
      port.echo("Test");
   }
}
```

Report a bug[201]

14.7.1.3.4. WSAuthorizationHandler

14.7.1.3.4.1. FQN:

org.picketlink.trust.jbossws.handler.WSAuthorizationHandler

Report a bug[202]

14.7.1.3.4.2. Objective:

Provide authorization capabilities to POJO based web services.

Report a bug[203]

14.7.1.3.4.3. Example Usage:

Please refer to the documentation on WSAuthenticationHandler.

Important

The example is in *WSAuthenticationHandler*[204] section.

[201] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29977-620583+%5BLatest%5D&comment=Title%3A+Example+Usage%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29977-620583+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[202] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29978-620584+%5BLatest%5D&comment=Title%3A+FQN%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29978-620584+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[203] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29979-620585+%5BLatest%5D&comment=Title%3A+Objective%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29979-620585+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[204] https://docs.jboss.org/author/display/PLINK/WSAuthenticationHandler

Report a bug[205]

14.7.1.4. Protecting EJB Endpoints

14.7.1.4.1. Introduction

PicketLink provides ways to protect your EJB endpoints using a SAML Security Token Service. This means that you can apply some security to your EJBs where only users with a valid SAML assertion can invoke to them.

This scenario is very common if you are looking for:

1. Leverage your Single Sign-On infrastructure to your service layer (EJBs, Web Services, etc)
2. Integrate your SAML Service Providers with your services by trusting the assertion previously issued by the Identity Provider
3. Any situation that requires the propagation of authorization/authentication information from one domain to another

Report a bug[206]

[205] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29980-626697+%5BLatest%5D&comment=Title%3A+Example+Usage%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29980-626697+01+Apr+2014+16%3A07+en-US+%5BLatest%5D

[206] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29981-620587+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.7.1.4.1.1. Process Overview

The client must first obtain the SAML assertion from PicketLink STS by sending a WS-Trust request to the token service. This process usually involves authentication of the client. After obtaining the SAML assertion from the STS, the client includes the assertion in the security context of the EJB request before invoking an operation on the bean. Upon receiving the invocation, the EJB container extracts the assertion and validates it by sending a WS-Trust validate message to the STS. If the assertion is considered valid by the STS (and the proof of possession token has been verified if needed), the client is authenticated.

Figure 14.1. TODO Gliffy image title empty

On JBoss, the SAML assertion validation process is handled by the SAML2STSLoginModule. It reads properties from a configurable file (specified by the configFile option) and establishes communication with the STS based on these properties. We will see how a configuration file looks like later on. If the assertion is valid, a Principal is created using the assertion subject name and if the assertion contains roles, these roles are also extracted and associated with the caller's Subject.

The client must first obtain the SAML assertion from the PicketLink STS or you Identity Provider. This process usually involves authentication of the client. After obtaining the SAML assertion, the client includes the assertion in the security context of the EJB request before invoking an operation on the bean. Upon receiving the invocation, the EJB container extracts the assertion and validates it by sending a WS-Trust validate message to the STS. If the assertion is considered valid by the STS (and the proof of possession token has been verified if needed), the client is authenticated.

On JBoss, the SAML assertion validation process is handled by the *Section 14.7.1.5.3, "SAML2STSLoginModule"* . It reads properties from a configurable file (specified by the configFile option) and establishes communication with the STS based on these properties. We will see how a configuration file looks like later on. If the assertion is valid, a Principal is created using the assertion subject name and if the assertion contains roles, these roles are also extracted and associated with the caller's Subject.

Report a bug[207]

14.7.1.4.2. Configuration

This section will cover two possible scenarios to protect and access your secured EJB endpoints. The main difference between these two scenarios is where the EJB client is deployed.

- Remote EJB Client using JNDI
- EJB Client is deployed at the same instance than your EJB endpoints

Report a bug[208]

14.7.1.4.2.1. Remote EJB Client using JNDI

Important

Before starting, please take a look at the following documentation *Remote EJB invocations via JNDI*[209] .

The configuration described in this section only works with versions 7.2.0+ and 7.1.3+ of JBoss Application Server.

If your endpoints are accessible from remote clients (in a different VM or server than your endpoints) you need to configure your JBoss Application Server 7 to allow use a SAML Assertion during the InitialContext creation.

Basically, the configuration envolves the following steps:

1. Add a new Security Realm to your standalone.xml
2. Create a Security Domain using the *Section 14.7.1.5.3, "SAML2STSLoginModule"*
3. Change the Remoting Connector to use the new Security Realm

[207] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29982-620703+%5BLatest%5D&comment=Title%3A+Process+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29982-620703+12+Mar+2014+12%3A37+en-US+%5BLatest%5D

[208] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29983-620589+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[209] https://docs.jboss.org/author/display/AS71/Remote+EJB+invocations+via+JNDI+-+EJB+client+API+or+remote-naming+project

Report a bug[210]

14.7.1.4.2.1.1. Add a new Security Realm

Important

Security Realms[211] are better described in the JBoss Application Server Documentation.

Edit your standalone.xml and add the following configuration for a new Security Realm:

```
<security-realm name="SAMLRealm">
    <authentication>
        <jaas name="ejb-remoting-sts"/>
    </authentication>
</security-realm>
```

The configuration above defines a Security Realm that delegates the username/password information to a JAAS Security Domain (that we'll create later) in order to authenticate an user.

When using the JAAS configuration for Security Realms, the remoting subsystem enables the PLAIN SASL authentication. This will allow your remote clients send the username/password where the password would be the previously issued SAML Assertion.In our case, the password will be the String representation of the SAML Assertion.

Tip

Make sure you also enable SSL. Otherwise all communication with the server will be done using plain text.

Report a bug[212]

14.7.1.4.2.1.2. Create a Security Domain using the SAML2STSLoginModule

Edit your standalone.xml and add the following configuration for a new Security Domain:

```
<security-domain name="ejb-remoting-sts" cache-type="default">
    <authentication>
```

[210] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29984-626696+%5BLatest%5D&comment=Title%3A+Remote+EJB+Client+using+JNDI%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[211] https://docs.jboss.org/author/display/AS71/Security+Realms

[212] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29985-626695+%5BLatest%5D&comment=Title%3A+Add+a+new+Security+Realm%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29985-626695+01+Apr+2014+16%3A06+en-US+%5BLatest%5D

```
        <login-module
 code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2STSLoginModule"
 flag="required" module="org.picketlink">
            <module-option name="configFile" value="${jboss.server.config.dir}/sts-
config.properties"/>
            <module-option name="password-stacking" value="useFirstPass"/>
        </login-module>
    </authentication>
</security-domain>
```

This configuration above defines a Security Domain that uses the SAML2STSLoginModule to get the String representation of the SAML Assertion and validate it against the Security Token Service.

You may notice that we provided a properties file as module-option. This properties file defines all the configuration needed to invoke the PicketLink STS. It should look like this:

```
serviceName=PicketLinkSTS
portName=PicketLinkSTSPort
endpointAddress=http://localhost:8080/picketlink-sts/PicketLinkSTS
username=admin
#password=admin
password=MASK-0BbleBL2LZk=
salt=18273645
iterationCount=56

#java -cp picketlink-fed-core.jar org.picketlink.identity.federation.core.util.PBEUtils
 18273645 56 admin
#Encoded password: MASK-0BbleBL2LZk=
```

This security domain will be used to authenticate your remote clients during the creation of the JNDI Initial Context.

Report a bug[213]

14.7.1.4.2.1.3. Change the Remoting Connector Security Realm

Edit your standalone.xml and change the security-realm attribute of the remoting connector:

```
<subsystem xmlns="urn:jboss:domain:remoting:1.1">
    <connector name="remoting-connector" socket-binding="remoting" security-
realm="SAMLRealm"/>
</subsystem>
```

The connector configuration is already present in your standalone.xml. You only need to change the security-realm attribute to match the one we created before.

Report a bug[214]

[213] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29986-620592+%5BLatest%5D&comment=Title%3A+Create+a+Security+Domain+using+the+SAML2STSLoginModule%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29986-620592+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[214] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29987-620593+%5BLatest%5D&comment=Title%3A+Change+the+Remoting+Connector+Security+Realm%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29987-620593+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

14.7.1.4.2.1.4. EJB Remote Client

The code above shows you how a EJB Rremote Client may look like:

```
// add the JDK SASL Provider that allows to use the PLAIN SASL Client
Security.addProvider(new Provider());

Element assertion = getAssertionFromSTS("UserA", "PassA");

// JNDI environment configuration properties
Hashtable<String, Object> env = new Hashtable<String, Object>();

env.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
env.put("java.naming.factory.initial",
 "org.jboss.naming.remote.client.InitialContextFactory");
env.put("java.naming.provider.url", "remote://localhost:4447");
env.put("jboss.naming.client.ejb.context", "true");
env.put("jboss.naming.client.connect.options.org.xnio.Options.SASL_POLICY_NOPLAINTEXT",
 "false");
env.put("javax.security.sasl.policy.noplaintext", "false");

// provide the user principal and credential. The credential is the previously issued SAML
 assertion
env.put(Context.SECURITY_PRINCIPAL, "admin");
env.put(Context.SECURITY_CREDENTIALS, DocumentUtil.getNodeAsString(assertion));

// create the JNDI Context and perform the authentication using the SAML2STSLoginModule
Context context = new InitialContext(env);

// lookup the EJB
EchoService object = (EchoService) context.lookup("ejb-test/EchoServiceImpl!
org.picketlink.test.trust.ejb.EchoService");

// If everything is ok the Principal name will be added to the message
Assert.assertEquals("Hi UserA", object.echo("Hi "));
```

Report a bug[215]

14.7.1.4.3. References

- JBoss AS 5 : *https://community.jboss.org/wiki/SAMLEJBIntegrationWithPicketLinkSTS*

Report a bug[216]

14.7.1.5. STS Login Modules

This page references the PicketLink Login Modules for the Security Token Server.

Report a bug[217]

[215] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29988-620594+%5BLatest%5D&comment=Title%3A+EJB+Remote+Client%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29988-620594+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[216] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29989-626694+%5BLatest%5D&comment=Title%3A+References%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29989-626694+01+Apr+2014+16%3A06+en-US+%5BLatest%5D

[217] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29990-620596+%5BLatest%5D&comment=Title%3A+STS+Login+Modules%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.7.1.5.1. References

Tip

PicketLink STS Login Modules[218] has the required details.

Report a bug[219]

14.7.1.5.2. JBWSTokenIssuingLoginModule

14.7.1.5.2.1. Fully Qualified Name

org.picketlink.trust.jbossws.jaas. **JBWSTokenIssuingLoginModule**

Report a bug[220]

14.7.1.5.2.2. Objective

A variant of the PicketLink STSIssuingLoginModule that allows us to:

1. Inject BinaryTokenHandler or SAML2Handler or both as client side handlers to the STS WS call.
2. Inject the JaasSecurityDomainServerSocketFactory DomainSocketFactory as a request property to the BindingProvider set to the key "org.jboss.ws.socketFactory". This is useful for mutually authenticated SSL with the STS where in we use a trust store defined by a JaasSecurityDomain instance.

Report a bug[221]

14.7.1.5.2.3. Configuration

Options Include:

- **configFile** : a properties file that gives details on the STS to the login module. This can be optional if you want to specify values directly.
- **handlerChain** : Comma separated list of handlers you need to set for handling outgoing message to STS. Values: binary (to inject BinaryTokenHandler), saml2 (to inject SAML2Handler), map (to inject MapBasedTokenHandler) or class name of your own handler with default constructor.

[218] http://community.jboss.org/wiki/PicketLinkSTSLoginModules
[219] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29991-626693+%5BLatest%5D&comment=Title%3A+References%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29991-626693+01+Apr+2014+16%3A05+en-US+%5BLatest%5D
[220] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29992-620598+%5BLatest%5D&comment=Title%3A+Fully+Qualified+Name%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29992-620598+12+Mar+2014+12%3A34+en-US+%5BLatest%5D
[221] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29993-620599+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29993-620599+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

- **cache.invalidation** : set it to "true" if you want the JBoss auth cache to invalidate caches based on saml token expiry. By default, this value is false.
- **inject.callerprincipa** l: set it to "true" if the login module should add a group principal called "CallerPrincipal" to the subject. This is useful in JBoss AS for programmatic security in web/ejb components.
- **groupPrincipalName** : by default, JBoss AS security uses "Roles" as the group principal name in the subject. You can give a different value.
- **endpointAddress** : endpoint url of STS
- **serviceName** : service Name of STS
- **portName** : port name of STS
- **username** : username of account on STS.
- **password** : password of account on STS
- **wsaIssuer** : if you need to customize the WS-Addressing Issuer address in the WS-Trust call to the STS.
- **wspAppliesTo** : if you need to customize the WS-Policy AppliesTo in the WS-Trust call to the STS.
- **securityDomainForFactory** : if you have a JaasSecurityDomain mbean service in JBoss AS that provides the truststore.
- **map.token.key** : key to find binary token in JAAS sharedState map. Defaults to "ClientID".
- **soapBinding** : allow to change SOAP binding for SAML reuest.
- **requestType** : allows to override SAML request type when sending request to STS. Default: "http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue" Other possible value: "http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate".

Note: The configFile option is optional. If you provide that, then it should be as below.

Configuration file such as sts-client.properties.

serviceName=PicketLinkSTS

portName=PicketLinkSTSPort

endpointAddress=http://localhost:8080/picketlink-sts/PicketLinkSTS

username=admin

password=admin

wsaIssuer=http://localhost:8080/someissuer

wspAppliesTo=http://localhost:8080/testws

Note:

- the password can be masked according to *http://community.jboss.org/wiki/PicketLinkConfigurationMaskpassword* which would give us something like, password=MASK-dsfdsfdslkfh

- wsaIssuer can be **optionally** added if you want a value for the WS-Addressing issuer in the WS-Trust call to the STS.
- wspAppliesTo can be **optionally** added if you want a value for WS-Policy AppliesTo in the WS-Trust call to the STS.
- serviceName, portName, endpointAddress are **mandatory** .
- username and password keys are not needed if you are using mutual authenticated ssl (MASSL) with the STS.

Report a bug[222]

14.7.1.5.2.3.1. SSL DomainSocketFactory in use by the client side

Many a times, the login module has to communicate with the STS over a mutually authenticated SSL. In this case, you want to specify the truststore. JBoss AS provides JaasSecurityDomain mbean to specify truststore. For this reason, there is a special JaasSecurityDomainServerSocketFactory that can be used for making the JBWS calls. Specify the "securityDomainForFactory" module option with the security domain name (in the JaasSecurityDomain mbean service).

Report a bug[223]

14.7.1.5.2.4. Example configurations

Either you specify the module options directly or you can use a properties file for the STS related properties.

Report a bug[224]

14.7.1.5.2.4.1. Configuration specified directly

```
<application-policy name="saml-issue-token">
 <authentication>
  <login-module
 code="org.picketlink.identity.federation.core.wstrust.auth.JBWSTokenIssuingLoginModule"
 flag="required">

       <module-option name="password-stacking">useFirstPass</module-option>

       <module-option name="endpointAddress">http://somests</module-option>

        <module-option name="serviceName">PicketLinkSTS</module-option>

       <module-option name="portName">PicketLinkPort</module-option>
```

[222] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29994-626692+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[223] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29995-620601+%5BLatest%5D&comment=Title%3A+SSL+DomainSocketFactory+in+use+by+the+client+side%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29995-620601+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[224] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A29996-620602+%5BLatest%5D&comment=Title%3A+Example+configurations%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
       <module-option name="username">admin</module-option>

       <module-option name="password">admin</module-option>

       <module-option name="inject.callerprincipal">true</module-option>
       <module-option name="groupPrincipalName">Membership</module-option>
     </login-module>
    </authentication>
</application-policy>
```

Report a bug[225]

14.7.1.5.2.4.2. Configuration with configFileOption

```
<application-policy name="saml-issue-token">
 <authentication>
  <login-module
 code="org.picketlink.identity.federation.core.wstrust.auth.JBWSTokenIssuingLoginModule"
 flag="required">
       <module-option name="configFile">/sts-client.properties</module-option>
       <module-option name="password-stacking">useFirstPass</module-option>
<module-option name="cache.invalidation">true</module-option>
<module-option name="inject.callerprincipal">true</module-option>
<module-option name="groupPrincipalName">Membership</module-option>
     </login-module>
    </authentication>
</application-policy>
```

Report a bug[226]

14.7.1.5.2.4.3. Dealing with Roles

If the STS sends roles via Attribute Statements in the SAML assertion, then the user has to use the SAMLRoleLoginModule.

```
<application-policy name="saml">
  <authentication>
    <login-module code="org.picketlink.trust.jbossws.jaas.JBWSTokenIssuingLoginModule"
 flag="required">
        <module-option name="endpointAddress">SOME_URL</module-option>
        <module-option name="serviceName">SecurityTokenService</module-option>
        <module-option name="portName">RequestSecurityToken</module-option>
        <module-option name="inject.callerprincipal">true</module-option>
        <module-option name="handlerChain">binary</module-option>
    </login-module>
    <login-module code="org.picketlink.trust.jbossws.jaas.SAMLRoleLoginModule"
 flag="required"/>
  </authentication>
</application-policy>
```

[225] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29997-620603+%5BLatest%5D&comment=Title%3A+Configuration+specified+directly%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29997-620603+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[226] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29998-620604+%5BLatest%5D&comment=Title%3A+Configuration+with+configFileOption%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29998-620604+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

If the STS does not send roles, then the user has to configure a different JAAS login module to pick the roles for the username. Something like the UsernamePasswordLoginModule.

```
<application-policy xmlns="urn:jboss:security-beans:1.0" name="binary">
      <authentication>
         <login-module code="org.picketlink.trust.jbossws.jaas.JBWSTokenIssuingLoginModule"
 flag="required">
            <module-option name="endpointAddress">http://localhost:8080/picketlink-sts/
PicketLinkSTS</module-option>
            <module-option name="serviceName">PicketLinkSTS</module-option>
            <module-option name="portName">PicketLinkSTSPort</module-option>
            <module-option name="inject.callerprincipal">true</module-option>
            <module-option name="handlerChain">binary</module-option>
            <module-option name="username">admin</module-option>
            <module-option name="password">MASK-0BbleBL2LZk=</module-option>
            <module-option name="salt">18273645</module-option>
            <module-option name="iterationCount">56</module-option>
            <module-option name="useOptionsCredentials">true</module-option>
            <module-option name="overrideDispatch">true</module-option>
            <module-option name="wspAppliesTo">http://services.testcorp.org/provider1</
module-option>
            <module-option name="wsaIssuer">http://something</module-option>
            <module-option name="password-stacking">useFirstPass</module-option>
         </login-module>

         <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
 flag="required">
            <module-option name="usersProperties">sts-users.properties</module-option>
            <module-option name="rolesProperties">sts-roles.properties</module-option>
            <module-option name="password-stacking">useFirstPass</module-option>
         </login-module>
      </authentication>
   </application-policy>
```

Report a bug[227]

14.7.1.5.3. SAML2STSLoginModule

14.7.1.5.3.1. FQN

org.picketlink.identity.federation.bindings.jboss.auth. **SAML2STSLoginModule**

Report a bug[228]

14.7.1.5.3.2. Author:

Stefan Guilhen

Report a bug[229]

[227] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+29999-620605+%5BLatest%5D&comment=Title%3A+Dealing+with+Roles%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=29999-620605+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[228] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30000-620606+%5BLatest%5D&comment=Title%3A+FQN%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30000-620606+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[229] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30001-620607+%5BLatest

14.7.1.5.3.3. Objective

This LoginModule authenticates clients by validating their SAML assertions with an external security token service (such as PicketLinkSTS). If the supplied assertion contains roles, these roles are extracted and included in the Group returned by the getRoleSets method.

The LoginModule could be also used to retrieve and validate SAML assertion token from HTTP request header.

Report a bug[230]

14.7.1.5.3.4. Module Options

This module defines the following module options:

- **configFile** - this property identifies the properties file that will be used to establish communication with the external security token service.
- **cache.invalidation** : set it to true if you require invalidation of JBoss Auth Cache at SAML Principal expiration.
- **jboss.security.security_domain** -security domain at which Principal will expire if cache.invalidation is used.
- **roleKey** : key of the attribute name that we need to use for Roles from the SAML assertion. This can be a comma-separated string values such as (Role,Membership)
- **localValidation** : if you want to validate the assertion locally for signature and expiry
- **localValidationSecurityDomain** : the security domain for the trust store information (via the JaasSecurityDomain)
- **tokenEncodingType** : encoding type of SAML token delivered via http request's header. Possible values are:
  - base64 - content encoded as base64. In case of encoding will vary between base64 and gzip use base64 and LoginModule will detect gzipped data.
  - gzip - gzipped content encoded as base64
  - none - content not encoded in any way
- **samlTokenHttpHeader** - name of http request header to fetch SAML token from. For example: "Authorize"
- **samlTokenHttpHeaderRegEx** - Java regular expression to be used to get SAML token from "samlTokenHttpHeader". Example: use: . **"(.**)".* to parse SAML token from header content like this: SAML_assertion="HHDHS=", at the same time set samlTokenHttpHeaderRegExGroup to 1.
- **samlTokenHttpHeaderRegExGroup** - Group value to be used when parsing out value of http request header specified by "samlTokenHttpHeader" using "samlTokenHttpHeaderRegEx".

%5D&comment=Title%3A+Author%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30001-620607+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[230] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30002-620608+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30002-620608+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

```
pattern = Pattern.compile(samlTokenHttpHeaderRegEx, Pattern.DOTALL);
Matcher m = pattern.matcher(content);
m.matches();
m.group(samlTokenHttpHeaderRegExGroup)
```

Any properties specified besides the above properties are assumed to be used to configure how the STSClient will connect to the STS. For example, the JBossWS StubExt.PROPERTY_SOCKET_FACTORY can be specified in order to inform the socket factory that must be used to connect to the STS. All properties will be set in the request context of the Dispatch instance used by the STSClient to send requests to the STS.

An example of a configFile can be seen bellow:

```
serviceName=PicketLinkSTS
portName=PicketLinkSTSPort
endpointAddress=[http://localhost:8080/picketlink-sts/PicketLinkSTS]
username=JBoss
password=JBoss
```

The first three properties specify the STS endpoint URL, service name, and port name. The last two properties specify the username and password that are to be used by the application server to authenticate to the STS and have the SAML assertions validated.

NOTE: Sub-classes can use getSTSClient() method to customize the STSClient class to make calls to STS

Report a bug[231]

14.7.1.5.3.5. Examples

Example Configuration 1:

```
<application-policy xmlns="urn:jboss:security-beans:1.0" name="cache-test">
      <authentication>
         <login-module
 code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2STSLoginModule"
 flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <module-option name="configFile">sts-config.properties</module-option>
            <module-option name="cache.invalidation">true</module-option>
            <module-option name="localValidation">true</module-option>
            <module-option name="localValidationSecurityDomain">MASSL</module-option>
         </login-module>
      </authentication>
   </application-policy>
```

Example Configuration 2 using http header and local validation:

```
   <application-policy xmlns="urn:jboss:security-beans:1.0" name="service">
      <authentication>
```

[231] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30003-620609+%5BLatest%5D&comment=Title%3A+Module+Options%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30003-620609+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

```
         <login-module
 code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2STSLoginModule"
 flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <module-option name="cache.invalidation">true</module-option>
            <module-option name="localValidation">true</module-option>
            <module-option name="localValidationSecurityDomain">java:jaas/
localValidationDomain</module-option>
            <module-option name="tokenEncodingType">gzip</module-option>
            <module-option name="samlTokenHttpHeader">Auth</module-option>
            <module-option name="samlTokenHttpHeaderRegEx">.*"(.*)".*</module-option>
            <module-option name="samlTokenHttpHeaderRegExGroup">1</module-option>
         </login-module>
         <login-module code="org.picketlink.trust.jbossws.jaas.SAMLRoleLoginModule"
 flag="required"/>
      </authentication>
   </application-policy>
```

In case of local validation here is example of jboss-beans.xml file to use to configure JAAS Security Domain for (JBoss AS6 or EAP5):

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="urn:jboss:bean-deployer:2.0">
   <!-- localValidationDomain bean -->
   <bean name="LocalValidationBean" class="org.jboss.security.plugins.JaasSecurityDomain">
     <constructor>
          <parameter>localValidationDomain</parameter>
     </constructor>
     <property name="keyStoreURL">file://${jboss.server.home.dir}/conf/stspub.jks</property>
     <property name="keyStorePass">keypass</property>
     <property name="keyStoreAlias">sts</property>
     <property name="securityManagement"><inject bean="JNDIBasedSecurityManagement"/></
property>
   </bean>
</deployment>
```

For JBoss AS7 or JBoss EAP6 add following security domain to your configuration file:

```

<security-domain name="localValidationDomain">
      <jsse
         keystore-password="keypass"
         keystore-type="JKS"
         keystore-url="file:///${jboss.server.config.dir}/stspub.jks"
         server-alias="sts"/>
</security-domain>
```

and reference this security domain as: <module-option name="localValidationSecurityDomain">localValidationDomain</module-option>.

Report a bug[232]

14.7.1.5.4. SAMLTokenCertValidatingLoginModule

org.picketlink.identity.federation.bindings.jboss.auth. **SAMLTokenCertValidatingLoginModule**

[232] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30004-620610+%5BLatest%5D&comment=Title%3A+Examples%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30004-620610+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

Report a bug[233]

14.7.1.5.4.1. Author:

Peter Skopek

Report a bug[234]

14.7.1.5.4.2. Objective

This LoginModule authenticates clients by validating their SAML assertions locally. If the supplied assertion contains roles, these roles are extracted and included in the Group returned by the getRoleSets method.

The LoginModule is designed to validate SAML token using X509 certificate stored in XML signature within SAML assertion token.

It validates:

1. CertPath against specified truststore. It has to have common valid public certificate in the trusted entries.
2. X509 certificate stored in SAML token didn't expire
3. if signature itself is valid
4. SAML token expiration

Report a bug[235]

14.7.1.5.4.3. Module Options

This module defines the following module options:

- **roleKey** : key of the attribute name that we need to use for Roles from the SAML assertion. This can be a comma-separated string values such as (Role,Membership)
- **localValidationSecurityDomain** : the security domain for the trust store information (via the JaasSecurityDomain)
- **cache.invalidation** - set it to true if you require invalidation of JBoss Auth Cache at SAML Principal expiration.
- **jboss.security.security_domain** -security domain at which Principal will expire if cache.invalidation is used.

[233] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A30005-620611+%5BLatest%5D&comment=Title%3A+SAMLTokenCertValidatingLoginModule%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[234] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30006-620612+%5BLatest%5D&comment=Title%3A+Author%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30006-620612+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[235] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30007-620613+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30007-620613+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

- **tokenEncodingType** : encoding type of SAML token delivered via http request's header. Possible values are:
  - base64 - content encoded as base64. In case of encoding will vary between base64 and gzip use base64 and LoginModule will detect gzipped data.
  - gzip - gzipped content encoded as base64
  - none - content not encoded in any way
- **samlTokenHttpHeader** - name of http request header to fetch SAML token from. For example: "Authorize"
- **samlTokenHttpHeaderRegEx** - Java regular expression to be used to get SAML token from "samlTokenHttpHeader". Example: use: . **"(.**)".* to parse SAML token from header content like this: SAML_assertion="HHDHS=", at the same time set samlTokenHttpHeaderRegExGroup to 1.
- **samlTokenHttpHeaderRegExGroup** - Group value to be used when parsing out value of http request header specified by "samlTokenHttpHeader" using "samlTokenHttpHeaderRegEx".

```
pattern = Pattern.compile(samlTokenHttpHeaderRegEx, Pattern.DOTALL);
Matcher m = pattern.matcher(content);
m.matches();
m.group(samlTokenHttpHeaderRegExGroup)
```

Report a bug[236]

14.7.1.5.4.4. Examples

Example Configuration 1:

```
<application-policy xmlns="urn:jboss:security-beans:1.0" name="certpath">
      <authentication>
         <login-module
 code="org.picketlink.identity.federation.bindings.jboss.auth.SAMLTokenCertValidatingLoginModule"
 flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <module-option name="cache.invalidation">true</module-option>
            <module-option name="localValidationSecurityDomain">java:jaas/
localValidationDomain</module-option>
         </login-module>
      </authentication>
</application-policy>
```

Example Configuration 2 using http header:

```
   <application-policy xmlns="urn:jboss:security-beans:1.0" name="service">
      <authentication>
         <login-module
 code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2STSLoginModule"
 flag="required">
            <module-option name="password-stacking">useFirstPass</module-option>
            <module-option name="cache.invalidation">true</module-option>
```

[236] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30008-620614+%5BLatest%5D&comment=Title%3A+Module+Options%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30008-620614+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

```
            <module-option name="localValidationSecurityDomain">java:jaas/
localValidationDomain</module-option>
            <module-option name="tokenEncodingType">gzip</module-option>
            <module-option name="samlTokenHttpHeader">Auth</module-option>
            <module-option name="samlTokenHttpHeaderRegEx">.*"(.*)".*</module-option>
            <module-option name="samlTokenHttpHeaderRegExGroup">1</module-option>
         </login-module>
      </authentication>
   </application-policy>
```

Example of jboss-beans.xml file to use to configure JAAS Security Domain containing trust store for above examples:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="urn:jboss:bean-deployer:2.0">
   <!-- localValidationDomain bean -->
   <bean name="LocalValidationBean" class="org.jboss.security.plugins.JaasSecurityDomain">
     <constructor>
          <parameter>localValidationDomain</parameter>
     </constructor>
     <property name="keyStoreURL">file://${jboss.server.home.dir}/conf/stspub.jks</property>
     <property name="keyStorePass">keypass</property>
     <property name="keyStoreAlias">sts</property>
     <property name="securityManagement"><inject bean="JNDIBasedSecurityManagement"/></
property>
   </bean>
</deployment>
```

Report a bug[237]

14.7.1.5.5. STSValidatingLoginModule

14.7.1.5.5.1. FQN:

org.picketlink.identity.federation.core.wstrust.auth.STSValidatingLoginModule

Report a bug[238]

14.7.1.5.5.2. Author:

Daniel Bevenius

Report a bug[239]

14.7.1.5.5.3. Objective/Features:

- Calls the configured STS and validates an available security token.

[237] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30009-620615+%5BLatest%5D&comment=Title%3A+Examples%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30009-620615+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[238] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30010-620616+%5BLatest%5D&comment=Title%3A+FQN%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30010-620616+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

[239] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30011-620617+%5BLatest%5D&comment=Title%3A+Author%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30011-620617+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

- A call to STS typically requires authentication. This LoginModule uses credentials from one of the following sources:
  - Its properties file, if the *useOptionsCredentials* module-option is set to true
  - Previous login module credentials if the *password-stacking* module-option is set to useFirstPass
  - From the configured *CallbackHandler* by supplying a *Name* and *Password Callback*
- Upon successful authentication, the SamlCredential is inserted in the Subject's public credentials if one with the same Assertion is not found to be already present there.
- New features included since 1.0.4 based on *PLFED-87*[240] :
  - If a Principal MappingProvider is configured, retrieves and inserts the Principal into the Subject
  - If a RoleGroup MappingProvider is configured, retrieves and inserts the user roles into the Subject
  - Roles can only be returned if they are included in the Security Token. Configure your STS to return roles through an AttributeProvider

Report a bug[241]

14.8. Extensions

14.8.1. Extensions

This page shows all the extensions and customizations available in the PicketLink project.

Report a bug[242]

14.8.2. PicketLinkAuthenticator

14.8.2.1. PicketLinkAuthenticator

14.8.2.1.1. FQN

org.picketlink.identity.federation.bindings.tomcat.PicketLinkAuthenticator

Report a bug[243]

[240] https://jira.jboss.org/browse/PLFED-87
[241] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30012-626690+%5BLatest%5D&comment=Title%3A+Objective%2FFeatures%3A%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30012-626690+01+Apr+2014+16%3A04+en-US+%5BLatest%5D
[242] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30013-620619+%5BLatest%5D&comment=Title%3A+Extensions%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30013-620619+12+Mar+2014+12%3A34+en-US+%5BLatest%5D
[243] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30014-620620+%5BLatest%5D&comment=Title%3A+FQN%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30014-620620+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

14.8.2.1.2. Objective

An authenticator that delegates actual authentication to a realm, and in turn to a security manager, by presenting a "conventional" identity. The security manager must accept the conventional identity and generate the real identity for the authenticated principal.

Report a bug[244]

14.8.2.1.3. JBoss Application Server 7.x Configuration

Your web.xml will define some security constraints. But it will define a <login-config> that is different from the servlet specifcation mandated BASIC, CLIENT-CERT, FORM or DIGEST methods. We suggest the use of SECURITY_DOMAIN as the method.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Restricted Access - Get Only</web-resource-name>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
  <role-name>STSClient</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>

<security-role>
    <role-name>STSClient</role-name>
</security-role>

<login-config>
    <auth-method>SECURITY_DOMAIN</auth-method>
    <realm-name>SECURITY_DOMAIN</realm-name>
    <form-login-config>
 <form-login-page>/login.html</form-login-page>
 <form-error-page>/error.html</form-error-page>
    </form-login-config>
</login-config>
```

Important

Note that we defined two pages in the **<form-login-config>** : **login.html** and **error.html** . Both pages must exists inside your deployment.

Change your WEB-INF/jboss-web.xml to configure the *PicketLinkAuthenticator* as a valve:

```
<jboss-web>
 <security-domain>authenticator</security-domain>
 <context-root>authenticator</context-root>
```

[244] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30015-620621+%5BLatest%5D&comment=Title%3A+Objective%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30015-620621+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

```
 <valve>
  <class-name>org.picketlink.identity.federation.bindings.tomcat.PicketLinkAuthenticator
  </class-name>
 </valve>
</jboss-web>
```

We also defined a **<security-domain>** configuration with the name of the security domain that you configured in your standalone.xml:

```
<security-domain name="authenticator" cache-type="default">
    <authentication>
        <login-module
 code="org.picketlink.test.trust.loginmodules.TestRequestUserLoginModule" flag="required">
            <module-option name="usersProperties" value="users.properties"/>
            <module-option name="rolesProperties" value="roles.properties"/>
        </login-module>
    </authentication>
</security-domain>
```

Tip

To use PicketLink you need to define it as a module dependency using the META-INF/jboss-deployment-structure.xml.

Report a bug[245]

14.8.2.1.4. JBoss Application Server 5.x Configuration

Your web.xml will define some security constraints. But it will define a <login-config> that is different from the servlet specifcation mandated BASIC, CLIENT-CERT, FORM or DIGEST methods. We suggest the use of SECURITY-DOMAIN as the method.

Create a context.xml in your WEB-INF directory of your web-archive.

```
<Context>
  <Valve
 className="org.picketlink.identity.federation.bindings.tomcat.PicketLinkAuthenticator" />
</Context>
```

Your web.xml may look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
   xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

   <description>Sales Application</description>
```

[245] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30016-620622+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+7.x+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30016-620622+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

```
   <security-constraint>
      <display-name>Restricted</display-name>
      <web-resource-collection>
         <web-resource-name>Restricted Access</web-resource-name>
         <url-pattern>/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
         <role-name>Sales</role-name>
      </auth-constraint>
      <user-data-constraint>
         <transport-guarantee>NONE</transport-guarantee>
      </user-data-constraint>
   </security-constraint>

   <security-role>
      <role-name>Sales</role-name>
   </security-role>

   <login-config>
      <auth-method>SECURITY-DOMAIN</auth-method>
   </login-config>
</web-app>
```

Warning

NOTE: The use of SECURITY-DOMAIN as the auth-method.

The war should be packaged as a regular web archive.

Report a bug[246]

14.8.2.1.4.1. Default Configuration at Global Level

If you have a large number of web applications and it is not practical to include context.xml in all the war files, then you can configure the "authenticators" attribute in the war-deployers-jboss-beans.xml file in /server/default/deployers/jbossweb.deployer/META-INF of your JBoss AS instance.

```
<property name="authenticators">
         <map class="java.util.Properties" keyClass="java.lang.String"
 valueClass="java.lang.String">
            <entry>
               <key>BASIC</key>
               <value>org.apache.catalina.authenticator.BasicAuthenticator</value>
            </entry>
            <entry>
               <key>CLIENT-CERT</key>
               <value>org.apache.catalina.authenticator.SSLAuthenticator</value>
            </entry>
            <entry>
               <key>DIGEST</key>
               <value>org.apache.catalina.authenticator.DigestAuthenticator</value>
```

[246] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A30017-620623+%5BLatest%5D&comment=Title%3A+JBoss+Application+Server+5.x+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

```
            </entry>
            <entry>
               <key>FORM</key>
               <value>org.apache.catalina.authenticator.FormAuthenticator</value>
            </entry>
            <entry>
               <key>NONE</key>
               <value>org.apache.catalina.authenticator.NonLoginAuthenticator</value>
            </entry>
               <key>SECURITY-DOMAIN</key>

<value>org.picketlink.identity.federation.bindings.tomcat.PicketLinkAuthenticator</value>
            </entry>

         </map>
      </property>
```

Report a bug[247]

14.8.2.1.4.2. Testing

1. Go to the deploy directory.
2. cp -R jmx-console.war test.war
3. In deploy/test.war/WEB-INF/web.xml, change the auth-method element to SECURITY-DOMAIN.
4.
```
<login-config>
      <auth-method>SECURITY-DOMAIN</auth-method>
      <realm-name>JBoss JMX Console</realm-name>
   </login-config>
```
5. Also uncomment the security constraints in web.xml. It should look as follows.
6.
```
<!-- A security constraint that restricts access to the HTML JMX console
   to users with the role JBossAdmin. Edit the roles to what you want and
   uncomment the WEB-INF/jboss-web.xml/security-domain element to enable
   secured access to the HTML JMX console.
   -->
   <security-constraint>
     <web-resource-collection>
       <web-resource-name>HtmlAdaptor</web-resource-name>
       <description>An example security config that only allows users with the
         role JBossAdmin to access the HTML JMX console web application
       </description>
       <url-pattern>/*</url-pattern>
       <http-method>GET</http-method>
       <http-method>POST</http-method>
     </web-resource-collection>
     <auth-constraint>
       <role-name>JBossAdmin</role-name>
     </auth-constraint>
   </security-constraint>
```
7. In the /server/default/conf/jboss-log4j.xml , add trace category for org.jboss.security.

[247] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30018-620624+%5BLatest%5D&comment=Title%3A+Default+Configuration+at+Global+Level%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30018-620624+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

8. Start JBoss AS.
9. Go to the following url: *http://localhost:8080/test/*
10. You should see a HTTP 403 message.
11. If you look inside the log, log/server.log, you will see the following exception trace:
12.

```
2011-04-20 11:02:01,714 TRACE
 [org.jboss.security.plugins.auth.JaasSecurityManagerBase.jmx-console]
 (http-127.0.0.1-8080-1) Login failure
javax.security.auth.login.FailedLoginException: Password Incorrect/Password Required
        at
 org.jboss.security.auth.spi.UsernamePasswordLoginModule.login(UsernamePasswordLoginModule.java:252)
        at
 org.jboss.security.auth.spi.UsersRolesLoginModule.login(UsersRolesLoginModule.java:152)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at
 sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at javax.security.auth.login.LoginContext.invoke(LoginContext.java:769)
        at javax.security.auth.login.LoginContext.access$000(LoginContext.java:186)
        at javax.security.auth.login.LoginContext$4.run(LoginContext.java:683)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.login.LoginContext.invokePriv(LoginContext.java:680)
        at javax.security.auth.login.LoginContext.login(LoginContext.java:579)
        at
 org.jboss.security.plugins.auth.JaasSecurityManagerBase.defaultLogin(JaasSecurityManagerBase.java:552)
        at
 org.jboss.security.plugins.auth.JaasSecurityManagerBase.authenticate(JaasSecurityManagerBase.java:486)
        at
 org.jboss.security.plugins.auth.JaasSecurityManagerBase.isValid(JaasSecurityManagerBase.java:365)
        at
 org.jboss.security.plugins.JaasSecurityManager.isValid(JaasSecurityManager.java:160)
        at
 org.jboss.web.tomcat.security.JBossWebRealm.authenticate(JBossWebRealm.java:384)
        at
 org.picketlink.identity.federation.bindings.tomcat.PicketLinkAuthenticator.authenticate(PicketLinkAuthentic
        at
 org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:491)
        at
 org.jboss.web.tomcat.security.JaccContextValve.invoke(JaccContextValve.java:92)
        at
 org.jboss.web.tomcat.security.SecurityContextEstablishmentValve.process(SecurityContextEstablishmentValve.j
        at
 org.jboss.web.tomcat.security.SecurityContextEstablishmentValve.invoke(SecurityContextEstablishmentValve.ja
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:127)
        at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:102)
        at
 org.jboss.web.tomcat.service.jca.CachedConnectionValve.invoke(CachedConnectionValve.java:158)
        at
 org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:109)
        at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:330)
        at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:829)
        at org.apache.coyote.http11.Http11Protocol
$Http11ConnectionHandler.process(Http11Protocol.java:598)
        at org.apache.tomcat.util.net.JIoEndpoint$Worker.run(JIoEndpoint.java:447)
        at java.lang.Thread.run(Thread.java:662)
```

As you can see from the stack trace, PicketLinkAuthenticator method has been kicked in.

Report a bug[248]

14.9. PicketLink API

14.9.1. Working with SAML Assertions

14.9.1.1. Introduction

This page shows you how to use the PicketLink API to programatically work with SAML Assertions.

The examples above demonstrates the following scenarios:

- How to parse a XML to a PicketLink AssertionType
- How to sign SAML Assertions
- How to validate SAML Assertions

The following API classes were used:

- org.picketlink.identity.federation.saml.v2.assertion.AssertionType
- org.picketlink.identity.federation.core.saml.v2.util.AssertionUtil
- org.picketlink.identity.federation.core.parsers.saml.SAMLParser
- org.picketlink.identity.federation.core.saml.v2.writers.SAMLAssertionWriter
- org.picketlink.identity.federation.api.saml.v2.sig.SAML2Signature
- org.picketlink.identity.federation.core.impl.KeyStoreKeyManager

Important

Please, check the javadoc for more informations about these classes.

Report a bug[249]

14.9.1.2. Parsing SAML Assertions

The PicketLink API provides the **org.picketlink.identity.federation.saml.v2.assertion.AssertionType** class to encapsulate the informations parsed from a SAML Assertion.

[248] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30019-626721+%5BLatest%5D&comment=Title%3A+Testing%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30019-626721+01+Apr+2014+16%3A21+en-US+%5BLatest%5D

[249] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30020-620626+%5BLatest%5D&comment=Title%3A+Introduction%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30020-620626+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

Let's suppose we have the following SAML Assertion:

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 ID="ID_75291c31-93f7-4f7f-8422-aacdb07466ee" IssueInstant="2012-05-25T10:40:58.912-03:00"
 Version="2.0">
    <saml:Issuer>http: //192.168.1.1:8080/idp-sig/</saml:Issuer>
    <saml:Subject>
        <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">user</
saml:NameID>
        <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml:SubjectConfirmationData InResponseTo="ID_326a389f-6a8a-4712-
b71d-77aa9c36795c" NotBefore="2012-05-25T10:40:58.894-03:00"
 NotOnOrAfter="2012-05-25T10:41:00.912-03:00" Recipient="http://192.168.1.4:8080/fake-sp" />
        </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2012-05-25T10:40:57.912-03:00"
 NotOnOrAfter="2012-05-25T10:41:00.912-03:00" />
    <saml:AuthnStatement AuthnInstant="2012-05-25T10:40:58.981-03:00">
        <saml:AuthnContext>
            <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</
saml:AuthnContextClassRef>
        </saml:AuthnContext>
    </saml:AuthnStatement>
    <saml:AttributeStatement>
        <saml:Attribute Name="Role">
            <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">test-role1</
saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="Role">
            <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">test-role2</
saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="Role">
            <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">test-role3</
saml:AttributeValue>
        </saml:Attribute>
    </saml:AttributeStatement>
</saml:Assertion>
```

The code to parse this XML is:

```
/**
     * <p>
     * Parses a SAML Assertion XML representation and convert it to a {@link AssertionType}
 instance.
     * </p>
     *
     * @throws Exception
     */
    @Test
    public void testParseAssertion() throws Exception {
        // get a InputStream from the source XML file
        InputStream samlAssertionInputStream = getSAMLAssertion();

        SAMLParser samlParser = new SAMLParser();

        Object parsedObject = samlParser.parse(samlAssertionInputStream);

        Assert.assertNotNull(parsedObject);
        Assert.assertTrue(parsedObject.getClass().equals(AssertionType.class));
```

```
        // cast the parsed object to the expected type, in this case AssertionType
        AssertionType assertionType = (AssertionType) parsedObject;

        // checks if the Assertion has expired.
        Assert.assertTrue(AssertionUtil.hasExpired(assertionType));

        // let's write the parsed assertion to the sysout
        ByteArrayOutputStream baos = new ByteArrayOutputStream();

        SAMLAssertionWriter writer = new
 SAMLAssertionWriter(StaxUtil.getXMLStreamWriter(baos));

        writer.write(assertionType);

        System.out.println(new String(baos.toByteArray()));
    }
```

Report a bug[250]

14.9.1.3. Signing a SAML Assertion

The PicketLink API provides the **org.picketlink.identity.federation.api.saml.v2.sig.SAML2Signature** to help during signature generation/validation for SAML Assertions.

```
/**
     * <p>
     * Signs a SAML Assertion.
     * </p>
     *
     * @throws Exception
     */
    @Test
    public void testSignAssertion() throws Exception {
        InputStream samlAssertionInputStream = getSAMLAssertion();

        // convert the InputStream to a DOM Document
        Document document = DocumentUtil.getDocument(samlAssertionInputStream);

        SAML2Signature samlSignature = new SAML2Signature();

        // get the key store manager instance.
        KeyStoreKeyManager keyStoreKeyManager = getKeyStoreManager();

        samlSignature.signSAMLDocument(document, keyStoreKeyManager.getSigningKeyPair());

        // let's print the signed assertion to the sysout
        System.out.println(DocumentUtil.asString(document));
    }
```

As you can see, we need to create a instance of **org.picketlink.identity.federation.core.impl.KeyStoreKeyManager** from where the certificates will be retrieved from. The code bellow shows you how to create it:

```
     /**
```

[250] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30021-620627+%5BLatest%5D&comment=Title%3A+Parsing+SAML+Assertions%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30021-620627+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

```
     * <p>
     * Creates a {@link KeyStoreKeyManager} instance.
     * </p>
     *
     * @throws Exception
     */
    private KeyStoreKeyManager getKeyStoreManager()
        throws TrustKeyConfigurationException, TrustKeyProcessingException {

        KeyStoreKeyManager keyStoreKeyManager = new KeyStoreKeyManager();

        ArrayList<AuthPropertyType> authProperties = new ArrayList<AuthPropertyType>();

        authProperties.add(createAuthProperty(KeyStoreKeyManager.KEYSTORE_URL,
 Thread.currentThread().getContextClassLoader().getResource("./keystore/
jbid_test_keystore.jks").getFile()));
        authProperties.add(createAuthProperty(KeyStoreKeyManager.KEYSTORE_PASS, "store123"));

        authProperties.add(createAuthProperty(KeyStoreKeyManager.SIGNING_KEY_ALIAS,
 "servercert"));
        authProperties.add(createAuthProperty(KeyStoreKeyManager.SIGNING_KEY_PASS,
 "test123"));

        keyStoreKeyManager.setAuthProperties(authProperties);

        return keyStoreKeyManager;
    }

    public AuthPropertyType createAuthProperty(String key, String value) {
        AuthPropertyType authProperty = new AuthPropertyType();

        authProperty.setKey(key);
        authProperty.setValue(value);

        return authProperty;
    }
```

Report a bug[251]

14.9.1.4. Validating a Signed SAML Assertion

The code to validate signatures is almost the same for signing. You still need a KeyStoreKeyManager instance.

```
/**
     * <p>
     * Validates a SAML Assertion.
     * </p>
     *
     * @throws Exception
     */
    @Test
    public void testValidateSignatureAssertion() throws Exception {
        InputStream samlAssertionInputStream = getSAMLSignedAssertion();

        KeyStoreKeyManager keyStoreKeyManager = getKeyStoreManager();
```

[251] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30022-620628+%5BLatest%5D&comment=Title%3A+Signing+a+SAML+Assertion%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30022-620628+12+Mar+2014+12%3A34+en-US+%5BLatest%5D

```
        Document signedDocument = DocumentUtil.getDocument(samlAssertionInputStream);

        boolean isValidSignature =
AssertionUtil.isSignatureValid(signedDocument.getDocumentElement(),
keyStoreKeyManager.getSigningKeyPair().getPublic());

        Assert.assertTrue(isValidSignature);
    }
```

Report a bug[252]

14.10. 3rd party integration

Common scenario is to use Picketlink as both Identity Provider (IDP) and Service Provider (SP), but sometimes it may be useful to integrate with 3rd party vendors as well. If your company is using services provided by 3rd party vendors like SalesForce or Google Apps, then SSO with these vendors may be real benefit for you.

We support these scenarios:

- *Picketlink as IDP, Salesforce as SP*[253]
- *Picketlink as IDP, Google Apps as SP*[254]
- *Picketlink as SP, Salesforce as IDP*[255]

Report a bug[256]

14.10.1. Picketlink as IDP, Salesforce as SP

In first scenario we will use Salesforce as SAML SP and we will use Picketlink application as SAML IDP. In this tutorial, we will reuse application **idp-sig.war** from *Picketlink quickstarts*[257] .

> NOTE: Integration is working from Picketlink version 2.1.2.Final and newer

Report a bug[258]

[252] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30023-620629+%5BLatest%5D&comment=Title%3A+Validating+a+Signed+SAML+Assertion%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30023-620629+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

[253] https://docs.jboss.org/author/display/PLINK/Picketlink+as+IDP%2C+Salesforce+as+SP

[254] https://docs.jboss.org/author/display/PLINK/Picketlink+as+IDP%2C+Google+Apps+as+SP

[255] https://docs.jboss.org/author/display/PLINK/Picketlink+as+SP%2C+Salesforce+as+IDP

[256] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A30024-626716+%5BLatest%5D&comment=Title%3A+3rd+party+integration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[257] https://docs.jboss.org/author/display/PLINK/PicketLink+Quickstarts#PicketLinkQuickstarts-AbouttheQuickstarts

[258] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A30025-626715+%5BLatest%5D&comment=Title%3A+Picketlink+as+IDP%2C+Salesforce+as+SP%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.10.1.1. Salesforce setup

Error

Topic 30026 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 30026* for more detailed information.

14.10.1.2. Picketlink IDP setup

- **Download and import Salesforce certificate** - SAMLRequest messages sent from Salesforce are signed with Salesforce certificate. In order to validate them, you need to download Salesforce client certificate from *http://wiki.developerforce.com/page/Client_Certificate* . Then you need to import the certificate into your keystore:

```
unzip -q /tmp/downloads/certificates/New_proxy.salesforce.com_certificate_chain.zip
keytool -import -keystore jbid_test_keystore.jks -file proxy-salesforce-com.123 -alias
 salesforce-cert
```

- **ValidatingAlias update** - You need to update ValidatingAlias section, so the SAMLRequest from Salesforce will be validated with Salesforce certificate. You need to add the line into file **idp-sig.war/WEB-INF/picketlink.xml** :

```
<ValidatingAlias Key="saml.salesforce.com" Value="salesforce-cert" />
```

- **Trusted domain** - update list of trusted domains and add domain "salesforce.com" to the list:

```
<Trust>
  <Domains>localhost,jboss.com,jboss.org,redhat.com,amazonaws.com,salesforce.com</Domains>
</Trust>
```

Report a bug[259]

14.10.1.2.1. Single logout

Error

Topic 30029 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 30029* for more detailed information.

[259] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A30027-626717+%5BLatest%5D&comment=Title%3A+Picketlink+IDP+setup%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.10.1.3. Test the setup

- Start the server with Picketlink IDP
- Visit URL of your salesforce domain. It should be likely something like: *https://yourdomain.my.salesforce.com/* . Now Salesforce will send SAMLRequest to your IDP and so you should be redirected to login screen on your IDP on *http://localhost:8080/idp-sig/*
- Login into Picketlink IDP as user *tomcat* . After successful login, SAMLRequest signature is validated by the certificate *salesforce-cert* and IDP produces SAMLResponse for IDP and performs redirection.
- Now Salesforce parse SAMLResponse, validates it signature with imported Picketlink certificate and then you should be redirected to salesforce and logged as user *tomcat* in your Salesforce domain.

Report a bug[260]

14.10.1.4. Troubleshooting

Salesforce provides simple tool in SSO menu, where you can see the status of last SAMLResponse sent to Salesforce SP and you can check what's wrong with the response here.

Good tool for checking communication between SP and IDP is also Firefox plugin *SAML Tracer*[261]

Report a bug[262]

14.10.2. Picketlink as SP, Salesforce as IDP

In this part, we will use Salesforce as IDP and sample application from Picketlink as SP.

> NOTE: Integration is working from Picketlink version 2.1.2.Final and newer

Report a bug[263]

[260] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30030-626720+%5BLatest%5D&comment=Title%3A+Test+the+setup%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30030-626720+01+Apr+2014+16%3A21+en-US+%5BLatest%5D

[261] https://addons.mozilla.org/en-US/firefox/addon/saml-tracer/

[262] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30031-626719+%5BLatest%5D&comment=Title%3A+Troubleshooting%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30031-626719+01+Apr+2014+16%3A20+en-US+%5BLatest%5D

[263] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A30032-620638+%5BLatest%5D&comment=Title%3A+Picketlink+as+SP%2C+Salesforce+as+IDP%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

14.10.2.1. Salesforce setup

Error

Topic 30033 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 30033* for more detailed information.

14.10.2.2. Picketlink Setup

As already mentioned, we will use sample application *sales-post-sig.war* from picketlink quickstarts.

- **Import salesforce IDP certificate** - In **sales-post-sig.war/WEB-INF/classes** you need to import downloaded certificate from salesforce into your keystore. You can use command like:

```
keytool -import -file salesforce_idp_cert.cer -keystore jbid_test_keystore.jks -alias
 salesforce-idp
```

- **Identity URL configuration** - In **sales-post-sig.war/WEB-INF/picketlink.xml** you need to change identity URL to something like:

```
<IdentityURL>${idp-sig.url::https://yourdomain.my.salesforce.com/idp/endpoint/HttpPost}
```

- **ValidatingAlias configuration** - In same file, you can add new validating alias for the salesforce host of your domain:

```
<ValidatingAlias Key="yourdomain.my.salesforce.com" Value="salesforce-idp" />
```

- **Roles mapping** - Last very important step is mapping of roles for users, which are logged through Salesforce IDP. Normally when you have Picketlink as both IDP and SP, then SAMLResponse from IDP usually contains *AttributeStatement* as part of SAML assertion and this statement contains list of roles in attribute *Role* . Picketlink SP is then able to parse list of roles from statement and then it leverages *SAML2LoginModule* to assign these roles to JAAS Subject of logged principal. Thing is that SAML Response from Salesforce IDP does not contain any attribute statement with roles, so you need to handle roles assignment by yourself. Easiest way could be to chain SAML2LoginModule with another login module (like *UsersRolesLoginModule* for instance), which will ensure that assigning of JAAS roles is delegated from SAML2LoginModule to the second Login Module in chain. Needed steps:
  - In **sales-post-sig.war/WEB-INF/jboss-web.xml** you can change security-domain from value *sp* to something different like *sp-salesforce*

```
<security-domain>sp-salesforce</security-domain>
```

- Create new application policy for this security domain. It differs in each application server, for example in JBoss 7 you need to edit **JBOSS_HOME/standalone/configuration/standalone.xml** and add this policy to particular section:

```
<security-domain name="sp-salesforce" cache-type="default">
  <authentication>
    <login-module
 code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2LoginModule"
 flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
    </login-module>
    <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
 flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="usersProperties" value="users.properties"/>
      <module-option name="rolesProperties" value="roles.properties"/>
    </login-module>
  </authentication>
</security-domain>
```

- In **sales-post-sig.war/WEB-INF/classes** you need to create empty file **users.properties** and non-empty file **roles.properties** where you need to map roles. For example you can add line like:

```
tomcat=manager,employee,sales
```

where **tomcat** is Federation ID of some user from Salesforce, which you will use for login.

Report a bug[264]

14.10.2.3. Test the integration

Now after server restart, let's try to access: *http://localhost:8080/sales-post-sig/* . You should be redirected to salesforce login page with SAMLRequest sent from your Picketlink sales-post-sig application. Now let's login into Salesforce with username and password of some Salesforce user from your domain (like tomcat user). Make sure that this user has Federation ID and this Federation ID is mapped in file roles.properties on Picketlink SP side like described in previous steps. Now you should be redirected to *http://localhost:8080/sales-post-sig/* as logged user.

Report a bug[265]

14.10.3. Picketlink as IDP, Google Apps as SP

Google Apps is another known business solution from Google. Google Apps supports SAML SSO in role of SAML SP, so you need to use your own application as SAML IDP. In this sample, we will again use *idp-sig.war* application from Picketlink quickstarts as IDP similarly like in *this tutorial*[266] .

> NOTE: Integration is working from Picketlink version 2.1.2.Final and newer

[264] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30034-620640+%5BLatest%5D&comment=Title%3A+Picketlink+Setup%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30034-620640+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

[265] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30035-626710+%5BLatest%5D&comment=Title%3A+Test+the+integration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30035-626710+01+Apr+2014+16%3A13+en-US+%5BLatest%5D

[266] https://docs.jboss.org/author/display/PLINK/Picketlink+as+IDP%2C+Salesforce+as+SP

Report a bug[267]

14.10.3.1. Google Apps setup

Error

Topic 30037 failed validation and is not included in this build.

Please review the compiler error for *Topic ID 30037* for more detailed information.

14.10.3.2. Picketlink IDP configuration

- **Trusted domains configuration** - update domains in **idp-sig.war/WEB-INF/picketlink.xml**

```
<Trust>
  <Domains>localhost,jboss.com,jboss.org,redhat.com,amazonaws.com,salesforce.com,google.com</
Domains>
</Trust>
```

- **Metadata configuration** - We don't want SAMLRequest from Google Apps to be validated, because it's not signed. So let's add another metadata for Google Apps, which will specify that SAMLRequest from Google Apps Service Provider won't be signed. So let's add another *EntityMetadataDescriptor* entry for your domain *google.com/a/yourdomain.com* into *sp-metadata.xml* file created in *previous tutorial*[268] (you may need to create new metadata file from scratch if not followed previous tutorial). Important attribute is especially *AuthnRequestsSigned* , which specifies that SAMLRequest from Google Apps are not signed.

```
<md:EntitiesDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
                       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                       xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata">
  <md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" entityID="https://
saml.salesforce.com" validUntil="2022-06-18T14:08:08.052Z">

......
  </md:EntityDescriptor>
   <md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" entityID="google.com/
a/yourdomain.com" validUntil="2022-06-13T21:46:02.496Z">
      <md:SPSSODescriptor AuthnRequestsSigned="false" WantAssertionsSigned="true"
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol" />
   </md:EntityDescriptor>
</md:EntitiesDescriptor>
```

Report a bug[269]

[267] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+IDs%3A%0A30036-626711+%5BLatest%5D&comment=Title%3A+Picketlink+as+IDP%2C+Google+Apps+as+SP%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A

[268] https://docs.jboss.org/author/display/PLINK/Picketlink+as+IDP%2C+Salesforce+as+SP

[269] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30038-626713+%5BLatest

14.10.3.3. Test it

Now logout from Google Apps and start server. And now you can do visit *https://mail.google.com/a/yourdomain.com* . After that Google Apps will send SAMLRequest and redirects you to *http://localhost:8080/idp-sig* . Please note that Google Apps is using SAML HTTP Redirect binding, so you can see SAMLRequest in browser URL. Also note that SAMLRequest is not signed, but this is not a problem as we configured it in metadata that requests from Google Apps are not signed. So after login into IDP as user tomcat, you should be automatically logged into your Google Apps as user "tomcat" as well.

Report a bug[270]

%5D&comment=Title%3A+Picketlink+IDP+configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30038-626713+01+Apr+2014+16%3A17+en-US+%5BLatest%5D

[270] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30039-626714+%5BLatest%5D&comment=Title%3A+Test+it%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30039-626714+01+Apr+2014+16%3A17+en-US+%5BLatest%5D

PicketLink Quickstarts

15.1. Overview

Quickstarts are self-contained, concise examples that generally demonstrate at most one or two features. The PicketLink quickstarts at GitHub provide working, buildable code that shows the usage of a number of authentication, authorization and identity management features. They are a nice way to communicate the design, common and best practices and the usage of PicketLink.

The PicketLink Quickstarts are part of the JBoss Developer Framework(*http://www.jboss.org/jdf/quickstarts/get-started/*[1]), formerly known as JDF. There you can find a lot of useful stuff about Java, JavaEE and of course all JBoss projects and products. Also, there are a lot of useful information and tutorials that will guide you to configure and prepare your environment to start using any of the available quickstarts.

All quickstarts are available at GitHub. So you can download them, fork the repository (if you have an GitHub account, of course) or even just clone the repository. The latter option is fully described along the README.md file for each quickstart, as well as a lot of additional information, configuration requirements, how to deploy and undeploy using the JBoss Enterprise Application Platform 6(and beyond) and so forth.

The repository is located at *https://github.com/jboss-developer/jboss-picketlink-quickstarts*[2].

Note

You don't need to be a Git expert in order to get the quickstarts. All the necessary commands are fully covered in the README.md file for each of them. For any additional information take a look at the JDF site too.

Report a bug[3]

15.2. Available Quickstarts

For a complete list of all available quickstarts, access the PicketLink Quickstart Repository.

Report a bug[4]

[1] http://www.jboss.org/jdf/quickstarts/get-started/
[2] https://github.com/jboss-developer/jboss-picketlink-quickstarts
[3] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30040-626704+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30040-626704+01+Apr+2014+16%3A10+en-US+%5BLatest%5D
[4] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30041-620647+%5BLatest%5D&comment=Title%3A+Available+Quickstarts%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30041-620647+12+Mar+2014+12%3A35+en-US+%5BLatest%5D

15.3. PicketLink Federation Quickstarts

We have a plenty of quickstarts covering some of the most important aspects of PicketLink Federation: SAML based SSO, WS-Trust support and so forth.

The PicketLink Federation quickstarts are prefixed with *picketlink-federation* (eg.: picketlink-federation-saml-idp-basic). Please, follow the steps described in the README.md file for each one.

Note

Users of PicketLink 2.1 series should start using the 2.5 version of the federation libraries. Once JBoss EAP is updated with the new module organization for 2.5 version, the 2.1 series will be deprecated.

Report a bug[5]

15.4. Contributing

PicketLink can be used to solve the most simple as well some of the most advanced security use cases. The main objective of the quickstarts is to cover most of use cases as we can, so people can quickly understand and start solving their own security-related problems based on what we're offering.

We would be very glad to have your contribution to our quickstarts list. If you have any suggestion about a new example, please create a JIRA and describe what you're looking for: *https://issues.jboss.org/browse/PLINK*[6].

You can also contribute by sending a Pull Request to the PicketLink Quickstarts repository which the code of your example application. Just follow the Contributing Guidelines from *http://site-jdf.rhcloud.com/quickstarts/get-involved/*[7].

Report a bug[8]

[5] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30042-665365+%5BLatest%5D&comment=Title%3A+PicketLink+Federation+Quickstarts%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30042-665365+07+Jun+2014+08%3A52+en-US+%5BLatest%5D

[6] https://issues.jboss.org/browse/PLINK

[7] http://site-jdf.rhcloud.com/quickstarts/get-involved/

[8] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+30043-626706+%5BLatest%5D&comment=Title%3A+Contributing%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=30043-626706+01+Apr+2014+16%3A12+en-US+%5BLatest%5D

Logging

16.1. Overview

During development you may need more runtime information in order to debug any unexpected behavior or perform some auditing. This section will show you how to enable logging to your application in order to be able to properly configure log messages and their respective levels.

PicketLink provides some specific logging categories, each of them related with a specific module or functionality:

Table 16.1. Logging Categories

Category	Scope	Description
`org.picketlink`	`GLOBAL`	Root category, used to enable logging for all PicketLink modules in use by your application.
`org.picketlink.idm`	`IDM`	PicketLink Identity Management root category.
`org.picketlink.idm.credential`	`IDM`	This category enables logging only for the **`Credential Management`** functionality provided by PicketLink Identity Management.
`org.picketlink.idm.identity`	`IDMstore`	This category enables logging only for the **`Identity Store`** in use by your application.
`org.picketlink.idm.identity`	`IDMstore.file`	This category enables logging only for the **`File Identity Store`**, if in use by your application.
`org.picketlink.idm.identity`	`IDMstore.jpa`	This category enables logging only for the **`JPA Identity Store`**, if in use by your application.
`org.picketlink.idm.identity`	`IDMstore.ldap`	This category enables logging only for the **`LDA[ Identity Store`**, if in use by your application.
`org.picketlink.authentication`	`BASE`	This category enables logging only for the **`Authentication`** functionality provided by the Base Module.
`org.picketlink.http`	`HTTP`	This category enables logging only for the **`Http Security`** functionality provided by the Base Module.

Report a bug[1]

16.2. Configuration

If you're using JBoss Enterprise Application Server or wildFly, just add the following configuration to the logging subsystem:

```
<!-- A file handler for PicketLink logging messages -->
<periodic-rotating-file-handler name="PICKETLINK">
    <file relative-to="jboss.server.log.dir" path="picketlink.log"/>
    <suffix value=".yyyy-MM-dd"/>
    <append value="true"/>
</periodic-rotating-file-handler>

<!-- This should enable logging for all PicketLink modules in use by your application -->
<logger category="org.picketlink">
    <level name="DEBUG"/>
    <handlers>
        <handler name="PICKETLINK"/>
    </handlers>
</logger>
```

The configuration above is very useful if you want to have a specific logging file for all PicketLink logging messages.

However, if you want to enable logging regardless the application server in use, you must have a *log4j.xml* file in your classpath as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">

  <!-- ============================== -->
  <!-- Append messages to the file -->
  <!-- ============================== -->
  <appender name="FILE" class="org.apache.log4j.DailyRollingFileAppender">
    <param name="File" value="${basedir}/target/test.log"/>
    <param name="Append" value="true"/>
    <param name="DatePattern" value="'.'yyyy-MM-dd"/>

    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%c] %m%n"/>
    </layout>
  </appender>


  <!-- ============================== -->
  <!-- Append messages to the console -->
  <!-- ============================== -->

  <appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.out"/>

    <layout class="org.apache.log4j.PatternLayout">
```

[1] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41553-702562+%5BLatest%5D&comment=Title%3A+Overview%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41553-702562+26+Aug+2014+00%3A18+en-US+%5BLatest%5D

```
      <param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%c] %m%n"/>
    </layout>
  </appender>

  <!-- ================ -->
  <!-- Limit categories -->
  <!-- ================ -->

  <!-- PicketLink Root category. -->
  <category name="org.picketlink">
    <priority value="DEBUG"/>
  </category>

  <!-- ====================== -->
  <!-- Setup the Root category -->
  <!-- ====================== -->

  <root>
    <appender-ref ref="CONSOLE"/>
    <appender-ref ref="FILE"/>
  </root>

</log4j:configuration>
```

You must also add the following dependency to your project:

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.6.0</version>
</dependency>
```

Report a bug[2]

[2] https://bugzilla.redhat.com/enter_bug.cgi?cf_environment=Build+Name%3A+22639%2C+PicketLink+Reference+Documentation-null%0ABuild+Date%3A+08-10-2014+05%3A49%3A25%0ATopic+ID%3A+41552-702567+%5BLatest%5D&comment=Title%3A+Configuration%0A%0ADescribe+the+issue%3A%0A%0A%0ASuggestions+for+improvement%3A%0A%0A%0AAdditional+information%3A&cf_build_id=41552-702567+26+Aug+2014+01%3A17+en-US+%5BLatest%5D

Compiler Output

Topic ID 35786

- INFO: JSON
- INFO: Assigned Writer: pedroigor Topic Types: Concept
- INFO: *Topic URL*[1]
- ERROR: This topic doesn't have well-formed xml. The element type "literal" must be terminated by the matching end-tag "</literal>". The processed XML is

```
<section>
 <title>JSON</title>
 <para>
  The JSON module is an optional module that implements and provides supports for the
 following specifications:
 </para>
 <itemizedlist>
  <listitem>
   <para>
    <literal>JWT</literal>, JSON Web Token
   </para>
  </listitem>
  <listitem>
   <para>
    <literal>JWS</literal>, JSON Web Signature
   </para>
  </listitem>
  <listitem>
   <para>
    <literal>JWK</literal>, JSON Web Key
   </para>
  </listitem>
  <listitem>
   <para>
    <literal>JWE</literal>, JSON Web Encryption
   </para>
  </listitem>
 </itemizedlist>
 <para>
  It provides an easy and type-safe API from where you can represent your own tokens
 following the specifications above. Or even consume tokens following these standards.
 </para>
 <para>
  This module is provided by the following library:
 </para>
 <itemizedlist>
  <listitem>
   <para>
    <literal>picketlink-json.jar
   </para>
  </listitem>
 </itemizedlist>
 <para>
  Documentation for this module is being prepared. For now, you can check some useful
 test cases that showcase how to use the API. Take a look here form more details, <link
 xlink:href="https://github.com/picketlink/picketlink/tree/master/modules/json/src/test/
java/org/picketlink/test/json/api">https://github.com/picketlink/picketlink/tree/master/
modules/json/src/test/java/org/picketlink/test/json/api</link>
```

[1] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=35786

```
 </para>
 <note>
  <para>
   This functionality is under development. The API is partially done, specially for JWT
 and JWS usages. Feedback and contributions are very welcome.
  </para>
 </note>
</section>
```

Topic ID 30033

- INFO: Salesforce setup
- INFO: *Topic URL*[2]
- ERROR: This topic has invalid DocBook XML. The error is *element "mediaobject" not allowed yet; expected element "info", "title" or "titleabbrev"*. The processed XML is

```
<section remap="TID_30033" version="5.0" xml:id="Salesforce_setup1" xmlns="http://
docbook.org/ns/docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Salesforce setup</title>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Disable Single Sign on</emphasis> in SSO settings if you
 enabled it previously. As in this step, we don't want to login into Salesforce through
 SSO but we want Salesforce to provide SSO for us and act as Identity Provider.
   </para>
  </listitem>
 </itemizedlist>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Identity provider setup</emphasis> - In link <emphasis
 role="strong">Setup</emphasis> -&gt; <emphasis role="strong">Security controls</emphasis>
 -&gt; <emphasis role="strong">Identity provider</emphasis> you need to setup Salesforce
 as IDP.
   </para>
  </listitem>
 </itemizedlist>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Generate certificate</emphasis> - first generate certificate
 on first screen. This certificate will be used to sign SAMLResponse messages sent from
 Salesforce IDP.
   </para>
   <figure>
    <mediaobject>
     <imageobject>
      <imagedata fileref="images/5880.png" />
     </imageobject>
    </mediaobject>
   </figure>
   <para>
    After certificate will be generated in Salesforce, you can download it to your
 computer.
   </para>
  </listitem>
  <listitem>
   <para>
```

[2] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=30033

```
    <emphasis role="strong">Configure generated certificate for Identity Provider</
emphasis> - In Identity Provider setup, you need to select the certificate, which you just
 generated
   </para>
  </listitem>
 </itemizedlist>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Add service provider</emphasis> - In section <emphasis
 role="strong">Setup</emphasis> -&gt; <emphasis role="strong">Security Controls</
emphasis> -&gt; <emphasis role="strong">Identity Provider</emphasis> -&gt; <emphasis
 role="strong">Service providers</emphasis> you can add your Picketlink application
 as Service Provider. We will use application <emphasis role="strong">sales-post-sig</
emphasis> from <link xlink:href="https://docs.jboss.org/author/display/PLINK/PicketLink
+Quickstarts#PicketLinkQuickstarts-AbouttheQuickstarts">Picketlink quickstarts</link> .
 So in first screen of configuration of your Service provider, you need to add <emphasis
 role="strong">ACS URL</emphasis> and <emphasis role="strong">Entity ID</emphasis>
 like <emphasis role="italics"> <link xlink:href="http://localhost:8080/sales-post-
sig/" /> </emphasis> <emphasis role="italics">. Subject type</emphasis> needs to be
 <emphasis role="italics">Federation ID</emphasis> and you also need to upload certificate
 corresponding to signing key of sales-post-sig application. You first need to export
 this certificate from your keystore file. See <link xlink:href="https://docs.jboss.org/
author/display/PLINK/Picketlink+as+IDP%2C+Salesforce+as+SP">previous tutorial</link> for
 how to do it. In next screen, you can select profile for users, who will be able to login
 to this Service Provider. By checking first checkbox, you will automatically select all
 profiles. After confirm this screen, you will have your service provider created. Let's
 see how your final configuration can looks like after confirming:
   </para>
   <figure>
    <mediaobject>
     <imageobject>
      <imagedata fileref="images/5881.png" />
     </imageobject>
    </mediaobject>
   </figure>
  </listitem>
 </itemizedlist>
 <sidebar>
  <para>
   <emphasis role="strong">WARNING:</emphasis> As mentioned in previous tutorial,
 you should create your own keystore file for Picketlink and not use example keystore
 <emphasis role="italics">jbid_test_keystore.jks</emphasis> and certificates from it
 in production environment. In this tutorial, we will use it only for simplicity and
 demonstration purposes.
  </para>
 </sidebar>
</section>
```

Topic ID 30037

- INFO: Google Apps setup
- INFO: *Topic URL*[3]
- ERROR: This topic has invalid DocBook XML. The error is *element "mediaobject" not allowed yet; expected element "info", "title" or "titleabbrev"*. The processed XML is

```
<section remap="TID_30037" version="5.0" xml:id="Google_Apps_setup" xmlns="http://
docbook.org/ns/docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Google Apps setup</title>
 <itemizedlist>
```

[3] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=30037

```
  <listitem>
   <para>
    <emphasis role="strong">Creating Google Apps domain</emphasis> - you need to create
 Google Apps domain on <link xlink:href="http://www.google.com/apps" /> . Follow the
 instructions on google page on how to do it.
   </para>
  </listitem>
  <listitem>
   <para>
    <emphasis role="strong">Add some users</emphasis> - let's add some users,
 which will be available to login into your domain. So let's add user <emphasis
 role="italics">tomcat</emphasis> first. In Google &amp; Apps control panel, you
 need to click <emphasis role="strong">Organization &amp; Users</emphasis> -&gt;
 <emphasis role="strong">Create new user</emphasis> and add him email <emphasis
 role="strong">tomcat@yourdomain.com</emphasis> . This will ensure that nick of new user
 will be <emphasis role="italics">tomcat</emphasis> . See screenshot:
   </para>
   <figure>
    <mediaobject>
     <imageobject>
      <imagedata fileref="images/5870.png" />
     </imageobject>
    </mediaobject>
   </figure>
  </listitem>
  <listitem>
   <para>
    <emphasis role="strong">Configure SAML SSO</emphasis> - In menu <emphasis
 role="strong">Advanced tools</emphasis> -&gt; <emphasis role="strong">Set up single
 sign-on (SSO)</emphasis> you can setup SSO settings. For our testing purposes, you
 can set it like done on screenshot . Especially it's important to set Sign-in page
 to <emphasis role="italics"> <link xlink:href="http://localhost:8080/idp-sig/" /> </
emphasis> <emphasis role="italics">. Sign-out page can be also set but Google Apps don't
 support SAML Single Logout profile, so this is only page where will be users redirected
 after logout. Let's click checkbox _Use a domain specific issuer</emphasis> to true.
   </para>
  </listitem>
 </itemizedlist>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Certificate upload</emphasis> - you also need to upload
 certificate exported from your picketlink keystore in similar way, like done for
 Salesforce in <link xlink:href="https://docs.jboss.org/author/display/PLINK/Picketlink
+as+IDP%2C+Salesforce+as+SP">previous tutorials</link> . So let's upload <emphasis
 role="italics">test-certificate.crt</emphasis> into Google Apps.
   </para>
  </listitem>
 </itemizedlist>
 <sidebar>
  <para>
   WARNING: Once again, you shouldn't use picketlink test keystore file
 jbid_test_keystore.jks in production environment. We use it here only for simplicity and
 for demonstration purposes.
  </para>
 </sidebar>
 <para>
  <figure>
   <mediaobject>
    <imageobject>
     <imagedata fileref="images/5869.png" />
    </imageobject>
   </mediaobject>
  </figure>
 </para>
</section>
```

Topic ID 30026

- INFO: Salesforce setup
- INFO: *Topic URL*[4]
- ERROR: This topic has invalid DocBook XML. The error is *element "mediaobject" not allowed yet; expected element "info", "title" or "titleabbrev"*. The processed XML is

```
<section remap="TID_30026" version="5.0" xml:id="Salesforce_setup" xmlns="http://
docbook.org/ns/docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Salesforce setup</title>
 <para>
  First you need to perform some actions on Salesforce side. Brief description is here.
 For more details, you can see Salesforce documentation.
 </para>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Register Salesforce account</emphasis> - You will
 need to register in Salesforce with free developer account. You can do it <link
 xlink:href="http://developer.force.com/">here</link> .
   </para>
  </listitem>
  <listitem>
   <para>
    <emphasis role="strong">Register your salesforce domain</emphasis> - Salesforce
 supports SP-initiated SAML login workflow or IDP-initiated SAML login workflow. For
 picketlink integration, we will use SP-initiated login workflow, where user needs
 to access Salesforce and Salesforce will send SAMLRequest to Picketlink IDP. For
 achieving this, you need to create Salesforce domain. When registered and logged in <link
 xlink:href="http://www.salesforce.com">www.salesforce.com</link> , you will need to click
 on Your name in right top corner -&gt; Link <emphasis role="strong">Setup</emphasis>
 -&gt; Link <emphasis role="strong">Company Profile</emphasis> -&gt; Link <emphasis
 role="strong">My Domain</emphasis> . Here you can create your Salesforce domain and make
 it available for testing.
   </para>
  </listitem>
  <listitem>
   <para>
    <emphasis role="strong">SAML SSO configuration</emphasis> - Now you need again
 to click on Your name in right top corner -&gt; Link <emphasis role="strong">Setup</
emphasis> -&gt; Link <emphasis role="strong">Security controls</emphasis> -&gt; Link
 <emphasis role="strong">Single Sign-On Settings</emphasis> Then configure it as follows:
   </para>
   <itemizedlist>
    <listitem>
     <para>
      <emphasis role="strong">SAML Enabled</emphasis> checkbox needs to be checked
     </para>
    </listitem>
    <listitem>
     <para>
      <emphasis role="strong">SAML Version</emphasis> needs to be 2.0
     </para>
    </listitem>
    <listitem>
     <para>
      <emphasis role="strong">Issuer</emphasis> needs to be <emphasis role="italics">
 <link xlink:href="http://localhost:8080/idp-sig/_">http://localhost:8080/idp-sig/</link>
 </emphasis> <emphasis role="italics">-</emphasis> This identifies issuer, which will be
 used as IDP for salesforce. NOTE: Be sure that URL really ends with "/" character.
```

[4] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=30026

```
    </para>
   </listitem>
   <listitem>
    <para>
     <emphasis role="strong">Identity Provider Login URL</emphasis> also needs to be
<link xlink:href="http://localhost:8080/idp-sig/_">http://localhost:8080/idp-sig/</link>
- This identifies URL where Salesforce SP will send it's SAMLRequest for login.
    </para>
   </listitem>
   <listitem>
    <para>
     <emphasis role="strong">Identity Provider Logout URL</emphasis> points to URL where
Salesforce redirects user after logout. You may also use your IDP address or something
different according to your needs.
    </para>
   </listitem>
   <listitem>
    <para>
     <emphasis role="strong">Subject mapping</emphasis> - You need to configure how to
map Subject from SAMLResponse, which will be send by Picketlink IDP, to Salesforce user
account. In the example, we will use that SAMLResponse will contain information about
Subject in "NameIdentifier" element of SAMLResponse and ID of subject will be mapped to
Salesforce Federation ID of particular user. So in: <emphasis role="strong">SAML User ID
Type</emphasis> , you need to check option <emphasis role="italics">Assertion contains
the Federation ID from the User object</emphasis> and for <emphasis role="strong">SAML
User ID Location</emphasis> , you need to check <emphasis role="italics">User ID is in
the NameIdentifier element of the Subject statement</emphasis> .
    </para>
   </listitem>
   <listitem>
    <para>
     <emphasis role="strong">Entity ID</emphasis> - Here we will use <link
xlink:href="https://saml.salesforce.com_">https://saml.salesforce.com</link> . Whole
configuration can look as follows:
    </para>
    <figure>
     <mediaobject>
      <imageobject>
       <imagedata fileref="images/5882.png" />
      </imageobject>
     </mediaobject>
    </figure>
   </listitem>
  </itemizedlist>
 </listitem>
</itemizedlist>
<itemizedlist>
 <listitem>
  <para>
   <emphasis role="strong">Certificate</emphasis> - Last very important thing is upload
of your certificate to Salesforce, because Salesforce needs to verify signature on
SAMLResponse sent from your Picketlink Identity Provider. So first you need to export
certificate from your keystore file and then import this certificate into Salesforce.
So in <emphasis role="strong">idp-sig.war/WEB-INF/classes</emphasis> you can run command
like:
  </para>
 </listitem>
</itemizedlist>
<informalexample>
 <programlisting>
keytool -export -keystore jbid_test_keystore.jks -alias servercert -file test-
certificate.crt
</programlisting>
 </informalexample>
 <para>
```

```
  after typing keystore password <emphasis role="italics">store123</emphasis> you should
 see exported certificate in file <emphasis role="italics">test-certificate.crt .</
emphasis>
 </para>
 <sidebar>
  <para>
   <emphasis role="strong">WARNING:</emphasis> For production environment in salesforce,
 you should generate your own keystore file and use certificate from your own file instead
 of the default picketlink <emphasis role="italics">jbid_test_keystore.jks</emphasis>
  </para>
 </sidebar>
 <para>
  Then you can import this certificate <emphasis role="italics">test-certificate.crt</
emphasis> into SalesForce via menu with SSO configuration.
 </para>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Adding users</emphasis> - Last action you need to do in
 Salesforce is to add some users. You can do it in: Link <emphasis role="strong">Setup</
emphasis> -&gt; Link <emphasis role="strong">Manage Users</emphasis> -&gt; Link <emphasis
 role="strong">Users</emphasis> . Now you can create user and fill some values as you
 want. Please note that username must be in form of email address. Note that <emphasis
 role="italics">Federation ID</emphasis> is the value, which is used for mapping the user
 with the NameIdentifier subject from SAML assertion, which will be sent from Picketlink
 IDP. So let's use Federation ID with value <emphasis role="italics">tomcat</emphasis> for
 our first user.
   </para>
  </listitem>
 </itemizedlist>
 <para>
  <figure>
   <mediaobject>
    <imageobject>
     <imagedata fileref="images/5884.png" />
    </imageobject>
   </mediaobject>
  </figure>
 </para>
</section>
```

Topic ID 30029

- INFO: Single logout
- INFO: *Topic URL*[5]
- ERROR: This topic has invalid DocBook XML. The error is *element "mediaobject" not allowed yet; expected element "info", "title" or "titleabbrev"*. The processed XML is

```
<section remap="TID_30029" version="5.0" xml:id="Single_logout" xmlns="http://docbook.org/
ns/docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Single logout</title>
 <para>
  Now you have basic setup done but in order to support single logout, you need to do some
 additional actions. Especially Salesforce is not using same URL for login and single
 logout, which means that we need to configure SP metadata on Picketlink side to provide
 mapping between SP and their URL for logout. Needed actions are:
 </para>
 <itemizedlist>
  <listitem>
   <para>
```

[5] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=30029

```
    <emphasis role="strong">Download SAML metadata</emphasis> from Salesforce SSO
 settings. Save downloaded XML file as <emphasis role="strong">idp-sig.war/WEB-INF/sp-
metadata.xml</emphasis>
   </para>
  </listitem>
 </itemizedlist>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Add SingleLogoutService element</emphasis> - unfortunately
 another element needs to be manually added into metadata as Salesforce doesn't use single
 logout configuration in their metadata. So let's add following element into metadata file
 after <emphasis role="italics">md:AssertionConsumerService</emphasis> element:
   </para>
  </listitem>
 </itemizedlist>
 <informalexample>
  <programlisting>
&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp; &amp;lt;md:SingleLogoutService
 Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;
 Location="https://login.salesforce.com/saml/logout-request.jsp?
saml=MgoTx78aEPkEM4eGV5ZzptlliwIVkRkOWYKlqXQq2StV_sLo0EiRqKYtIc" index="0"
 isDefault="true"/&amp;gt;
</programlisting>
 </informalexample>
 <para>
  Note that value of Location attribute will be different for your domain. You
 can see which value to use in Salesforce SSO settings page from element <emphasis
 role="italics">Salesforce.com Single Logout URL</emphasis> :
 </para>
 <para>
  <figure>
   <mediaobject>
    <imageobject>
     <imagedata fileref="images/5883.png" />
    </imageobject>
   </mediaobject>
  </figure>
 </para>
 <itemizedlist>
  <listitem>
   <para>
    <emphasis role="strong">Add md:EntitiesDescriptor element</emphasis> - Finally you
 need to add enclosing element <emphasis role="italics">md:EntitiesDescriptor</emphasis>
 and encapsulate whole current content into it. This is needed as we may want to use more
 EntityDescriptor elements in this single metadata file (like another element for Google
 Apps etc):
   </para>
  </listitem>
 </itemizedlist>
 <informalexample>
  <programlisting>
&amp;lt;?xml version="1.0" encoding="UTF-8"?&amp;gt;
&amp;lt;md:EntitiesDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;
 xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"&amp;gt;
&amp;nbsp; &amp;lt;md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
 entityID="https://saml.salesforce.com" ....
&amp;nbsp; ...
&amp;nbsp; &amp;lt;/md:EntityDescriptor&amp;gt;
&amp;lt;/md:EntitiesDescriptor&amp;gt;
</programlisting>
 </informalexample>
 <itemizedlist>
```

```
  <listitem>
   <para>
    <emphasis role="strong">Configure metadata location</emphasis> - Let's add new
 MetaDataProvider into file <emphasis role="strong">idp-sig.war/WEB-INF/picketlink.xml</
emphasis> after section with KeyProvider:
   </para>
  </listitem>
 </itemizedlist>
 <informalexample>
  <programlisting>
...
         &lt;/KeyProvider&gt;

         &lt;MetaDataProvider
 ClassName="org.picketlink.identity.federation.core.saml.md.providers.FileBasedEntitiesMetadataProvide
           &lt;Option Key="FileName" Value="/WEB-INF/sp-metadata.xml"/&gt;
         &lt;/MetaDataProvider&gt;
      &lt;/PicketLinkIDP&gt;
.....
</programlisting>
 </informalexample>
</section>
```

Topic ID 29879

- INFO: Example:
- INFO: *Topic URL*[6]
- ERROR: This topic has invalid DocBook XML. The error is *element "section" incomplete; expected element "address", "anchor", "annotation", "bibliolist", "blockquote", "bridgehead", "calloutlist", "caution", "classsynopsis", "cmdsynopsis", "constraintdef", "constructorsynopsis", "destructorsynopsis", "epigraph", "equation", "example", "fieldsynopsis", "figure", "formalpara", "funcsynopsis", "glosslist", "important", "indexterm", "info", "informalequation", "informalexample", "informalfigure", "informaltable", "itemizedlist", "literallayout", "mediaobject", "methodsynopsis", "msgset", "note", "orderedlist", "para", "procedure", "productionset", "programlisting", "programlistingco", "qandaset", "refentry", "remark", "revhistory", "screen", "screenco", "screenshot", "section", "segmentedlist", "sidebar", "simpara", "simplelist", "simplesect", "subtitle", "synopsis", "table", "task", "tip", "titleabbrev", "variablelist" or "warning"*. The processed XML is

```
<section remap="TID_29879" version="5.0" xml:id="Example14" xmlns="http://docbook.org/ns/
docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Example:</title>
</section>
```

Topic ID 29851

- INFO: Introduction
- INFO: *Topic URL*[7]
- ERROR: This topic has invalid DocBook XML. The error is *element "mediaobject" not allowed yet; expected element "info", "title" or "titleabbrev"*. The processed XML is

```
<section remap="TID_29851" version="5.0" xml:id="Introduction3" xmlns="http://docbook.org/
ns/docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Introduction</title>
```

[6] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=29879
[7] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=29851

```
<para>
 The PicketLink Identity Provider Authenticator is a component responsible for the
authentication of users and for issue and validate SAML assertions.
</para>
<para>
 Basically, there two different different authenticator implementations type:
</para>
<itemizedlist>
 <listitem>
  <para>
   Identity Provider Authenticators
  </para>
 </listitem>
 <listitem>
  <para>
   Service Provider Authenticators
  </para>
 </listitem>
</itemizedlist>
<para>
 <figure>
  <mediaobject>
   <imageobject>
    <imagedata fileref="images/5885.png" />
   </imageobject>
  </mediaobject>
 </figure>
</para>
</section>
```

Topic ID 29920

- INFO: Example:
- INFO: *Topic URL*[8]
- ERROR: This topic has invalid DocBook XML. The error is *element "section" incomplete; expected element "address", "anchor", "annotation", "bibliolist", "blockquote", "bridgehead", "calloutlist", "caution", "classsynopsis", "cmdsynopsis", "constraintdef", "constructorsynopsis", "destructorsynopsis", "epigraph", "equation", "example", "fieldsynopsis", "figure", "formalpara", "funcsynopsis", "glosslist", "important", "indexterm", "info", "informalequation", "informalexample", "informalfigure", "informaltable", "itemizedlist", "literallayout", "mediaobject", "methodsynopsis", "msgset", "note", "orderedlist", "para", "procedure", "productionset", "programlisting", "programlistingco", "qandaset", "refentry", "remark", "revhistory", "screen", "screenco", "screenshot", "section", "segmentedlist", "sidebar", "simpara", "simplelist", "simplesect", "subtitle", "synopsis", "table", "task", "tip", "titleabbrev", "variablelist" or "warning"*. The processed XML is

```
<section remap="TID_29920" version="5.0" xml:id="Example13" xmlns="http://docbook.org/ns/
docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Example:</title>
</section>
```

Topic ID 29894

- INFO: Example:
- INFO: *Topic URL*[9]

[8] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=29920
[9] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=29894

- ERROR: This topic has invalid DocBook XML. The error is *element "section" incomplete; expected element "address", "anchor", "annotation", "bibliolist", "blockquote", "bridgehead", "calloutlist", "caution", "classsynopsis", "cmdsynopsis", "constraintdef", "constructorsynopsis", "destructorsynopsis", "epigraph", "equation", "example", "fieldsynopsis", "figure", "formalpara", "funcsynopsis", "glosslist", "important", "indexterm", "info", "informalequation", "informalexample", "informalfigure", "informaltable", "itemizedlist", "literallayout", "mediaobject", "methodsynopsis", "msgset", "note", "orderedlist", "para", "procedure", "productionset", "programlisting", "programlistingco", "qandaset", "refentry", "remark", "revhistory", "screen", "screenco", "screenshot", "section", "segmentedlist", "sidebar", "simpara", "simplelist", "simplesect", "subtitle", "synopsis", "table", "task", "tip", "titleabbrev", "variablelist" or "warning"*. The processed XML is

```
<section remap="TID_29894" version="5.0" xml:id="Example2" xmlns="http://docbook.org/ns/
docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Example:</title>
</section>
```

Topic ID 29913

- INFO: Example:
- INFO: *Topic URL*[10]
- ERROR: This topic has invalid DocBook XML. The error is *element "section" incomplete; expected element "address", "anchor", "annotation", "bibliolist", "blockquote", "bridgehead", "calloutlist", "caution", "classsynopsis", "cmdsynopsis", "constraintdef", "constructorsynopsis", "destructorsynopsis", "epigraph", "equation", "example", "fieldsynopsis", "figure", "formalpara", "funcsynopsis", "glosslist", "important", "indexterm", "info", "informalequation", "informalexample", "informalfigure", "informaltable", "itemizedlist", "literallayout", "mediaobject", "methodsynopsis", "msgset", "note", "orderedlist", "para", "procedure", "productionset", "programlisting", "programlistingco", "qandaset", "refentry", "remark", "revhistory", "screen", "screenco", "screenshot", "section", "segmentedlist", "sidebar", "simpara", "simplelist", "simplesect", "subtitle", "synopsis", "table", "task", "tip", "titleabbrev", "variablelist" or "warning"*. The processed XML is

```
<section remap="TID_29913" version="5.0" xml:id="Example12" xmlns="http://docbook.org/ns/
docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Example:</title>
</section>
```

Topic ID 29813

- INFO: Introduction
- INFO: *Topic URL*[11]
- ERROR: This topic has invalid DocBook XML. The error is *element "mediaobject" not allowed yet; expected element "info", "title" or "titleabbrev"*. The processed XML is

```
<section remap="TID_29813" version="5.0" xml:id="Introduction2" xmlns="http://docbook.org/
ns/docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Introduction</title>
 <para>
  PicketLink Service Providers Authenticators are important components responsible for
 the authentication of users using the SAML Assertion previously issued by an Identity
 Provider.
```

[10] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=29913
[11] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=29813

```
 </para>
 <para>
  They are responsible for intercepting each request made to an application, checking
 if a SAML assertion is present in the request, validating its signature and executing
 SAML specific validations and creating a security context for the user in the requested
 application.
 </para>
 <para>
  <figure>
   <mediaobject>
    <imageobject>
     <imagedata fileref="images/5885.png" />
    </imageobject>
   </mediaobject>
  </figure>
 </para>
</section>
```

Topic ID 29775

- INFO: Introduction
- INFO: *Topic URL*[12]
- ERROR: This topic has invalid DocBook XML. The error is *element "mediaobject" not allowed yet; expected element "info", "title" or "titleabbrev"*. The processed XML is

```
<section remap="TID_29775" version="5.0" xml:id="Introduction5" xmlns="http://docbook.org/
ns/docbook" xmlns:xlink="http://www.w3.org/1999/xlink">
 <title>Introduction</title>
 <para>
  The PicketLink Identity Provider Authenticator is a component responsible for the
 authentication of users and for issue and validate SAML assertions.
 </para>
 <para>
  <figure>
   <mediaobject>
    <imageobject>
     <imagedata fileref="images/5885.png" />
    </imageobject>
   </mediaobject>
  </figure>
 </para>
</section>
```

Topic ID 29982

- INFO: Process Overview
- INFO: *Topic URL*[13]
- ERROR: No image filename specified. Must be in the format [ImageFileID].extension e.g. 123.png, or images/321.jpg

[12] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=29775
[13] http://virt-ecs-01.lab.eng.bne.redhat.com:8080/pressgang-ccms-ui/#SearchResultsAndTopicView;query;topicIds=29982

Compiler Glossary

"... is possibly an invalid custom Injection Point."	• The XML comment mentioned has been identified as a possible custom Injection Point, that is incorrectly referenced. • To fix this error please ensure that the type is valid, a colon is used to separate the IDs from the type and only topic IDs are used in the ID list.
"This topic contains an invalid element that can't be converted into a DOM Element."	• The topic XML contains invalid elements that cannot be successfully converted in DOM elements. • To fix this error please remove or correct any invalid XML elements or entities.
"This topic doesn't have well-formed xml."	• The topic XML is not well-formed XML and maybe missing opening or closing element statements. • To fix this error please ensure that all XML elements having an opening and closing statement and all XML reserved characters are represented as XML entities.
"This topic has invalid DocBook XML."	• The topic XML is not valid against the DocBook specification. • To fix this error please ensure that all XML elements are valid DocBook elements . Also check to ensure all XML sub elements are valid for the root XML element.
"This topic has invalid Injection Points."	• The topic XML contains Injection Points that cannot be resolved into links. • To fix this error please ensure that all the topics referred to by Injection Points are included in the build and/or have adequate relationships.
"This topic has no XML data"	• The topic doesn't have any XML Content to display. • To fix this warning, open the topic URL and add some content.

Appendix A. Revision History

Revision 1.0.0-1 **Wed Oct 08 2014** **CS Builder Robot** ***robot@dev.null.com***

Built from Content Specification: 22639, Revision: 715410