AuthzForce CE

Release 10.1.1

Cyril Dangerville, THALES

Aug 04, 2021

Table of Contents

This project is part of FIWARE. You will find more information about this FIWARE GE at the Catalogue.

CHAPTER 1

Introduction

AuthzForce is the reference implementation of the Authorization PDP Generic Enabler (formerly called Access Control GE). Indeed, as mandated by the GE specification, this implementation provides an API to get authorization decisions based on authorization policies, and authorization requests from PEPs. The API follows the REST architecture style, and complies with XACML v3.0. XACML (eXtensible Access Control Markup Language) is a OASIS standard for authorization policy format and evaluation logic, as well as for the authorization decision request/response format. The PDP (Policy Decision Point) and the PEP (Policy Enforcement Point) terms are defined in the XACML standard. This GEri plays the role of a PDP.

To fulfill the XACML architecture, you may need a PEP (Policy Enforcement Point) to protect your application, which is not provided here. For REST APIs, we can use the PEP Proxy (Wilma) available in the FIWARE catalogue.

CHAPTER 2

Release Notes

The release notes are available on our GitHub release page.

CHAPTER 3

Features

See the list of features on Github for the current version.

CHAPTER 4

Installation and Administration Guide

This guide provides the procedure to install the AuthzForce server, including system requirements and troubleshooting instructions.

4.1 System Requirements

The system requirements are the following:

- CPU frequency: 2.6 GHz min
- CPU architecture: i686/x86_64
- RAM: 4GB min
- Disk space: 10 GB min
- File system: ext4
- Operating System: Ubuntu 18.04 LTS or later
- Java environment:
  - JRE 11 either from OpenJDK or Oracle;
  - Tomcat 9.x.

4.2 Installation

If you are already using an older version of AuthzForce and wish to migrate your setup to the new version, please backup the folder `/opt/authzforce-ce-server` first because it will be overwritten by the new version, then proceed with the *Minimal setup* below, to install the new version; finally, proceed with the *Upgrade* section that follows, to transfer data from the old version.

4.2.1 Minimal setup

1. Install a JRE 11 if you don't have one already, using either of these two methods depending on your JDK preference:
   - For OpenJDK: `$ sudo apt install openjdk-11-jre`
2. Install Tomcat 9: `$ sudo apt install tomcat9`.
3. Each AuthzForce Server version number has the form MAJOR.MINOR.PATH (Semantic Versioning). Identify the latest binary (Ubuntu package with `.deb` extension) release of AuthzForce Server on Maven Central Repository that matches the MAJOR.MINOR version of this documentation. This is the current latest software version to which this documentation version applies. If you want to use a different software version, go to the latest documentation version with matching MAJOR.MINOR and follow instructions there. Else you may download the software version. We will refer to its version number as `M.m.P` (please replace accordingly):

   $ wget https://repo1.maven.org/maven2/org/ow2/authzforce/authzforce-ce-server-dist/M.m.P/authzforce-ce-server-dist-M.m.P.deb

   You should get a file called `authzforce-ce-server-dist-M.m.P.deb`.
4. Copy this file to the host where you want to install the software.
5. On the host, from the directory where you copied this file, run the following commands:

```
$ sudo apt install curl
$ sudo apt install ./authzforce-ce-server-dist-M.m.P.deb
```

6. At the end, you will see a message giving optional instructions to go through. Please follow them as necessary.

Note that Tomcat default configuration may specify a very low value for the Java `Xmx` flag, causing the AuthzForce webapp startup to fail. In that case, make sure Tomcat with `Xmx` at 1 Go or more (2 Go recommended). You can fix it as follows:

```
$ sudo sed -i "s/-Xmx128m/-Xmx1024m/" /etc/default/tomcat
$ sudo systemctl restart tomcat9
```

Known issue: lack of entropy may cause delays in Tomcat start up on virtual machines in particular: more info on Entropy Source issue. **So beware.**

4.2.2 Upgrade

If you are still using an older version of AuthzForce and wish to migrate your setup to the new version, assuming you made a backup in a separate location, as told previously, please follow these steps:

1. Download AuthzForce server upgrader distribution (.tar.gz extension) from Maven Central Repository in the same version as the Server version you want to upgrade to. You get a file called `authzforce-ce-server-upgrader-M.m.P.tar.gz` (replace `M.m.P` with the corresponding version).
2. Copy this file to the host where the old AuthzForce Server is installed, and unzip it and change directory:

```
$ tar xvzf authzforce-ce-server-upgrader-M.m.P.tar.gz
$ cd authzforce-ce-server-upgrader-M.m.P
```

3. Follow the instructions in file `README.html`.

4.2.3 Advanced setup

The *Minimal setup* gave you minimal installation steps to get started testing the features of the GE API. This may be enough for testing purposes, but barely for production. If you are targeting a production environment, you have to carry out extra installation and configuration steps to address non-functional aspects: security (including availability), performance, etc. The *Appendix* also gives some recommendations on what you should do.

4.3 Administration

4.3.1 Tomcat

For configuring and managing Tomcat, please refer to the official user guide.

4.3.2 AuthzForce webapp

The AuthzForce webapp configuration directory is located here: `/opt/authzforce-ce-server/conf`.

In particular, the file `logback.xml` configures the logging for the webapp (independently from Tomcat). By default, AuthzForce-specific logs go to `/var/log/tomcat9/authzforce-ce/error.log`.

Restart Tomcat to apply any configuration change:

```
$ sudo systemctl restart tomcat9
```

4.3.3 Fast Infoset mode

Fast Infoset is an ITU-T/ISO standard for representing XML (XML Information Set to be accurate) using binary encodings, designed for use cases to provide smaller encoding sizes and faster processing than a W3C XML representation as text. The open source Fast Infoset project provide some performance results and more information about the standardisation status. There are several use cases at the origin of Fast Infoset. A major one comes from the Web3D consortium that is responsible for open standards in real-time 3D communication, and that adopted Fast Infoset for the serialization and compression of X3D documents. X3D is a standard for representing 3D scenes and objects using XML.

AuthzForce Server offers experimental support for Fast Infoset (use with caution). This feature is disabled by default. To enable Fast Infoset support, change the value of the parameter `spring.profiles.active` to `+fastinfoset` in the webapp context configuration file `/etc/tomcat9/Catalina/localhost/authzforce-ce.xml`; then restart Tomcat as shown in the previous section in order to apply changes.

4.3.4 Policy Domain Administration

4.3.4.1 The Concept of Policy Domain

The application is multi-tenant, i.e. it allows users or organizations to work on authorization policies in complete isolation from each other. In this document, we use the term *domain* instead of *tenant*. In this context, a policy domain consists of:

- Various metadata about the domain: ID assigned by the AuthzForce API, external ID (assigned by the provisioning client), description;
- A policy repository;

- Attribute Providers configuration: attribute providers provide attributes that the PEP does NOT directly provide in the XACML <Request>. For example, an attribute provider may get attribute values from an external database.

The reasons for creating different domains:

- Users or organizations do not want others to access their data, or even be impacted by others working on the same application.
- The same user or organization may want to work on different domains for different use cases; e.g. work with one policy for production environment, another for testing, another for a specific use case project, etc.

4.3.4.2 Default Domain Settings

Administrators can set default settings for all domains to make sure domains are created in a proper configuration according to an administrative policy, or, in more simple terms, the administrator's preferences. The administrator may change these settings in the various XML files inside the folder `/opt/authzforce-ce-server/conf/domain.tmpl`:

- `pdp.xml`:
  - `maxVariableRefDepth`: optional, positive integer that indicates the maximum depth of Variable reference chaining allowed in policies: `VariableDefinition` 1 -> `VariableDefinition` 2 -> ..., where -> represents a XACML VariableReference. No limit if undefined. This property applies only to policies loaded by the PDP, i.e. the root policy and policies referenced from it directly or indirectly via XACML PolicySetIdReference.
  - `maxPolicyRefDepth`: optional, positive integer that indicates the maximum depth of Policy(Set) reference chaining: `PolicySet` 1 -> `PolicySet` 2 -> ... -> `PolicySet` N; where -> represents a XACML PolicySetIdReference. No limit if undefined. This property applies only to policies loaded by the PDP, i.e. the root policy and policies referenced from it directly or indirectly via XACML PolicySetIdReference.
  - `clientRequestErrorVerbosityLevel`: optional, positive integer (default: 0) that sets the level of detail in the XACML StatusDetail element returned in the Indeterminate Decision Result in case of bad Request (XACML syntax/content is invalid). Increasing this value usually helps better pinpoint the reason why a particular Request was rejected by the XACML parser. This only applies to the content of the HTTP request body (XACML), it does not apply to HTTP-level errors (e.g. bad HTTP headers), in which case you get a HTTP status code 400 without any XACML response since the request is rejected before the body is passed to the XACML parser.
- `policies/cm9vdA/0.1.0.xml`: the default root XACML PolicySet enforced by the PDP on the domain. As an administrator, you may change the content of this policy on two conditions:
  1. You **must not** change the `PolicySetId`.
  2. If you change the `Version` (e.g. to `1.2.3`), you **must** change the filename prefix (before `.xsd` extension) to the same value (e.g. `1.2.3.xsd`).
- `properties.xml`: other domain properties, more specifically:
  - `maxPolicyCount`: optional, stricly positive integer that indicates the maximum number of policies on a domain, no limit if undefined.
  - `maxVersionCountPerPolicy`: optional, stricly positive integer that indicates the maximum number of versions per policy, no limit if undefined.
  - `versionRollingEnabled`: boolean, true if and only if policy versions should be rolled over, i.e. when `maxVersionCountPerPolicy` has been reached, oldest versions are automatically removed to make place.

4.3.4.3 Domain Creation

You create a domain by doing a HTTP POST request with XML payload to URL: `http://${SERVER_NAME}:${PORT}/authzforce-ce/domains`. Replace `${SERVER_NAME}` and `${PORT}` with your server hostname and port for HTTP. You can do it with `curl` tool with the the following content in a XML file (`domainProperties.xml` in this example) as the HTTP request body:

```
$ cat domainProperties.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<domainProperties
 xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
 externalId="external0">
 <description>This is my domain</description>
</domainProperties>

$ curl --verbose --request "POST" \
--header "Content-Type: application/xml;charset=UTF-8" \
--data @domainProperties.xml \
--header "Accept: application/xml" \
  http://${SERVER_NAME}:${PORT}/authzforce-ce/domains

...
> POST /authzforce-ce/domains HTTP/1.1
> Content-Type: application/xml;charset=UTF-8
> Accept: application/xml
> Content-Length: 227
>
...
< HTTP/1.1 200 OK
< Server: Authorization System
< Date: Mon, 04 Aug 2016 13:00:12 GMT
< Content-Type: application/xml
< Transfer-Encoding: chunked
<
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<link xmlns="http://www.w3.org/2005/Atom"
  rel="item" href="h_D23LsDEeWFwqVFFMDLTQ"
  title="h_D23LsDEeWFwqVFFMDLTQ"/>
```

WARNING: Mind the leading and trailing single quotes for the `--data` argument. Do not use double quotes instead of these single quotes, otherwise curl will remove the double quotes in the XML payload itself, and send invalid XML which will be rejected by the server. You may use the `--trace-ascii -` argument (the last dash here means *stdout*) instead of `--verbose`, in order to check the actual request body sent by `curl`. So use it only if you need to dump the outgoing (and incoming) data, in particular the request body, on *stdout*.

The `href` value in the response above gives you the domain ID (in the form of a Base64-encoded UUID) assigned by the API. You need this ID for any further operation on the domain.

4.3.4.4 Domain Removal

You remove a domain by doing a HTTP DELETE request with XML payload to URL: `http://${SERVER_NAME}:${PORT}/authzforce-ce/domains/{domain_ID}`. For example with `curl` tool:

```
$ curl --verbose --request DELETE \
 --header "Content-Type: application/xml;charset=UTF-8" \
 --header "Accept: application/xml" \
 http://${SERVER_NAME}:${PORT}/authzforce-ce/domains/h_D23LsDEeWFwqVFFMDLTQ
```

Policy administration is part of the Authorization Server API, addressed more extensively in the *User and Programmers Guide*.

4.4 High Availability

In order to achieve high availability with multiple AuthzForce Server instances (AuthzForce Server cluster), you need to make sure that the following directories are synchronized on all instances:

- Configuration directory: `/opt/authzforce-ce-server/conf`. This directory is not modified by the API but only by administrators having access to the directory, and any change to it requires restarting Tomcat to apply. Therefore, this directory requires synchronization only after a manual change by a server admin, which should not occur very often. When it occurs, the server administrators may reproduce the changes on each instance manually; or, if there are too many instances for this to be practical, they may use automatic file synchronization solutions, or a distributed filesystems (e.g. NFS) combined with file monitoring solutions. Both kinds of solutions must be capable of executing a specific command, to restart Tomcat in this case, whenever a filesystem change in the directory is detected on a instance node. For example, csync2 is a solution of the first kind that is free and open source.
- Data directory: `/opt/authzforce-ce-server/data`. This is where the Server API persists and retrieves domain data such as policies. Therefore, it is critical to keep this directory synchronized across all the nodes in the high availability cluster, using either file synchronization solutions such as csync2, or distributed file systems such as NFS. Besides, for usability and performance reasons, the AuthzForce server caches certain objects in memory such as domains' PDPs and ID-externalId mappings (more info in the *User and Programmers Guide*). Therefore, it is also critical to re-sync the AuthzForce Server cache after certain changes done directly by aforementioned solutions to the local data directory. There are two ways to do that:
  - **REST API**: you can keep the server in sync with the data directory by calling the following API operations, dependending on the type of change:
    - HEAD `/domains`: to be used after any global change to the data directory. Inappropriate and largely suboptimal if there are many domains but changes concern only one or a few of them, in which case the next operations should be preferred.
    - HEAD `/domains/{domainId}`: to be used after a specific domain directory `/opt/authzforce-ce-server/data/domains/{domainId}` is created.
    - DELETE `/domains/{domainId}`: to be used after a specific domain directory `/opt/authzforce-ce-server/data/domains/{domainId}` is deleted.
    - HEAD `/domains/{domainId}/properties`: to be used after a specific domain's properties file `/opt/authzforce-ce-server/data/domains/{domainId}/properties.xml` is modified (especially the `externalId` property).
    - HEAD `/domains/{domainId}/pap/pdp.properties`: to be used after a specific domain's PDP configuration file `/opt/authzforce-ce-server/data/domains/{domainId}/pdp.xml` or policies directory `/opt/authzforce-ce-server/data/domains/{domainId}/policies` is modified.

    In these operations, you may use `GET` method instead of `HEAD` as well. However, `HEAD` is recommended for better performances as it does not return any content (response body), on the contrary to `GET`. Beware that the `Content-Length` returned by a `HEAD` is still the same as would be returned by the `GET` equivalent. In any case, if you opt for the file synchronization solution as mentioned earlier, you would have to make it call one of these operations depending on the type of change detected. If you opt for the distributed file system, you would need a file monitoring solution to detect changes and make such calls.
  - **Embedded file monitoring threads**: it is possible to enable file monitoring threads embedded in AuthzForce Server. These threads check for changes to the local data directory periodically, and synchronize the cache automatically. This feature is disabled by default. To enable it, change the value of the parameter `org.ow2.authzforce.domains.sync.interval` to a strictly positive integer in the webapp context configuration file `/etc/tomcat9/Catalina/localhost/authzforce-ce.xml`. The parameter value indicates the period between two checks for changes, in seconds. Beware that this feature creates one

extra thread per domain. Therefore, the impact on memory and CPU usage increases with the number of domains. Last but not least, **use this feature only on filesystems that support millisecond or higher resolution of file timestamps**, such as `ext4` (supports nanosecond resolution). Indeed, Authzforce file monitoring threads use file timestamps to detect changes. As a result, if the resolution of the filesystem is coarser than the millisecond, and a file change occurred in less than a second after the last check, it will go undetected (the file's *mtime* timestamp is not updated), and synchronization will not work as expected.

4.5 Sanity check procedures

The Sanity Check Procedures are the steps that a System Administrator will take to verify that the installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

4.5.1 End-to-End testing

To check the proper deployment and operation of the AuthzForce Server, perform the following steps:

1. Get the list of policy administration domains by doing the following HTTP request, replacing `${host}` with the server hostname, and `${port}` with the HTTP port of the server, for example with `curl` tool:

```
$ curl --verbose --show-error --write-out '\n' \
  --request GET http://${host}:${port}/authzforce-ce/domains
```

2. Check the response which should have the following headers and body (there may be more headers which do not require checking here):

```
Status Code: 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resources
  xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
  xmlns:atom="http://www.w3.org/2005/Atom">
  <atom:link rel="item" href="1XepFknrEea2mQAdYFsFBQ" title="1XepFknrEea2mQAdYFsFBQ"/>
  ... list of links to other policy domains omitted here...
</resources>
```

You can check the exact body format in the representation element of response code 200 for method `getDomains`, and all other API resources and operations in general, in the WADL (Web Application Description Language) document available at the following URL:

```
http://${host}:${port}/authzforce-ce/?_wadl
```

4.5.2 List of Running Processes

- One or more `java` processes for Tomcat.

4.5.3 Network interfaces Up & Open

- TCP 22;
- TCP 8080.

The port 8080 can be replaced by any other port Tomcat is listening to for HTTP connections to the webapp.

4.5.4 Databases

None.

4.6 Diagnosis Procedures

1. Perform the test described in **'End to End testing'_**.
2. If you get a Connection Refused/Error, check whether Tomcat is started:

```
$ sudo systemctl status tomcat9
```

3. If status stopped, start Tomcat:

```
$ sudo systemctl start tomcat9
```

4. If Tomcat fails to start, check for any Tomcat high-level error in Tomcat log directory: `/var/log/tomcat9`
5. If Tomcat is successfully started (no error in server logs), perform the test described in **'End to End testing'_** again.
6. If you still get a Connection Refused/error, check whether Tomcat is not listening on a different port:

```
$ sudo netstat -lataupen|grep java
```

7. If you still get a connection refused/error, especially if you are connecting remotely, check whether you are able to connect locally, then check the network link, i.e. whether any network filtering is in place on the host or on the access network, or other network issue: network interface status, DNS/IP adress resolution, routing, etc.
8. If you get an error `404 Not Found`, make sure the webapp is deployed and enabled in Tomcat. Check for any webapp deployment error in file: `/var/log/tomcat9/authzforce-ce/error.log`.

4.6.1 Resource availability

To have a healthy enabler, the resource requirements listed in *System Requirements* must be satisfied, in particular:

- Minimum RAM: 4GB;
- Minimum CPU: 2.6 GHz;
- Minimum Disk space: 10 GB.

4.6.2 Remote Service Access

None.

4.6.3 Resource consumption

The resource consumption strongly depends on the number of concurrent clients and requests per client, the number of policy domains (a.k.a. tenants in this context) managed by the Authorization Server, and the complexity of the policies defined by administrators of each domain.

The memory consumption shall remain under 80% of allocated RAM. See *System Requirements* for the minimum required RAM.

The CPU usage shall remain under 80% of allocated CPU. See *System Requirements* for the minimum required CPU.

As for disk usage, at any time, there should be 1GB free space left on the disk.

4.6.4 I/O flows

- HTTPS flows with possibly large XML payloads to port 8443 or whatever port Tomcat is listening to for HTTPS connections to the webapp;
- HTTP flows with possibly large XML payloads to port 8080 or whatever port Tomcat is listening to for HTTP connections to the webapp.

4.7 Appendix

4.7.1 Security setup for production

You have to secure the environment of the application server and the server itself. Securing the environment of a server in general will not be addressed here, because it is a large subject for which you can find a lot of public documentation. You will learn about perimeter security, network and transport-level security (firewall, IDS/IPS...), OS security, application-level security (Web Application Firewall), etc. For instance, the *NIST Guide to General Server Security* (SP 800-123) is a good start.

4.7.1.1 Server Security Setup

For more Tomcat-specific security guidelines, please read Tomcat 9 Security considerations.

For security of communications (confidentiality, integrity, client/server authentication), it is also recommended to enable SSL/TLS with PKI certificates. The first step to set up this is to have your Certification Authority (PKI) issue a server certificate for your AuthzForce instance. You can also issue certificates for clients if you want to require client certificate authentication to access the AuthzForce server/API. If you don't have such a CA at hand, you can create your own (a basic one) with instructions given in the next section.

4.7.1.2 Certificate Authority Setup

If you have a CA already, you can skip this section. So this section is about creating a basic local Certificate Authority (CA) for internal use. This CA will be in charge of issuing certificates to the Authorization Server and clients, for authentication, integrity and confidentiality purposes. **This procedure requires JDK 1.7 or later.** (For the sake of simplicity, we do not use a subordinate CA, although you should for production, see keytool command example, use the `pathlen` parameter to restrict number of subordinate CA, `pathlen=0` means no subordinate.)

1. Generate the CA keypair and certificate on the platform where the Authorization Server is to be deployed (change the validity argument to your security requirements, example here is 365 days):

```
$ keytool -genkeypair -keystore taz-ca-keystore.jks -alias taz-ca \
  -dname "CN=My Organization CA, O=FIWARE" -keyalg RSA -keysize 2048 \
  -validity 365 -ext bc:c="ca:true,pathlen:0"
```

2. Export the CA certificate to PEM format for easier distribution to clients:

```
$ keytool -keystore taz-ca-keystore.jks -alias taz-ca \
  -exportcert -rfc > taz-ca-cert.pem
```

4.7.1.3 Server SSL Certificate Setup

For Tomcat 9, refer to the Tomcat 9 SSL/TLS Configuration HOW-TO.

4.7.1.4 Web Application Secutity

4.7.1.4.1 XML and JSON Security

The AuthzForce web application exposes a XML-based API. Therefore it is vulnerable to XML denial-of-service attacks. To mitigate these attacks, there are two solutions:

- **AuthzForce native protection**: you can add the following Environment entries in AuthzForce webapp context file `/etc/tomcat9/Catalina/localhost/authzforce-ce.xml` (if an entry is absent or its value is negative, the default value is used):

```
<Environment
 name="org.apache.cxf.stax.maxChildElements"
 description="Maximum number of child elements (resp. properties) in an input XML element (resp. JSON)."
 type="java.lang.Integer"
 value="1000"
 override="false" />

<Environment
 name="org.apache.cxf.stax.maxElementDepth"
 description="Maximum depth of an element (resp. JSON object) in input XML (resp. JSON)."
 type="java.lang.Integer"
 value="100"
 override="false" />

<!--Following entries are not supported in Fast Infoset mode
(more info: https://issues.apache.org/jira/browse/CXF-6848) -->
<Environment
 name="org.apache.cxf.stax.maxAttributeCount"
 description="Maximum number of attributes per element in input XML."
 type="java.lang.Integer"
 value="100"
 override="false" />

<Environment
 name="org.apache.cxf.stax.maxAttributeSize"
 description="Maximum size of a single attribute in input XML."
 type="java.lang.Integer"
 value="1000"
 override="false" />

<Environment
 name="org.apache.cxf.stax.maxTextLength"
 description="Maximum size of XML text node (resp. JSON string)  in input XML (resp. JSON)."
 type="java.lang.Integer"
 value="1000"
 override="false" />

<Environment
 name="org.ow2.authzforce.domains.xacmlJsonSchemaRelativePath"
            description="Path to JSON schema file for XACML JSON Profile's Request validation, relative␣
↪to ${org.ow2.authzforce.config.dir} (if undefined/empty value, the Request.schema.json file from␣
↪authzforce-ce-xacml-json-model project is used by default). This property applies if and only if 'org.
↪ow2.authzforce.domains.enableXacmlJsonProfile' is true."
 type="java.lang.String"
 value=""
 override="false" />
```

  Restart Tomcat to apply changes.

- **Dedicated WAF**: for better mitigation, we recommend using a WAF (Web Application Firewall) with XML attack mitigation features in front of the Authzforce server.

There are commercial as well as open source WAFs available on the market. However, beware that this solution is not compatible with Fast Infoset, unless the WAF itself supports Fast Infoset. Similarly, if you want to use TLS, then the WAF or some proxy in front of it must support TLS to be the TLS server endpoint.

4.7.1.4.2 Disabling unused features

You can disable all PAP features, i.e. make the REST API read-only by setting the `enablePdpOnly` environment entry to `true` in AuthzForce webapp context file `/etc/tomcat9/Catalina/localhost/authzforce-ce.xml` (if an entry is absent or its value is negative, the default value is used):

```
<Environment name="org.ow2.authzforce.domains.enablePdpOnly" value="true" type="java.lang.
↪Boolean" override="false"
description="Enable PDP only, i.e. disable all PAP (or other administration) features iff true
↪" />
```

4.7.1.5 User and Role Management Setup

In production, access to the API must be restricted and explicitly authorized. To control which clients can do what on which resources, we need to have access to user identity and attributes and assign proper roles to them. These user and role management features are no longer supported by the AuthzForce server itself, but should be delegated to the Identity Management GE.

4.7.1.6 Domain Role Assignment

In production, access to the API must be restricted and explicitly authorized. To control which clients can do what on what parts of API, we need to have access to user identity and attributes and assign proper roles to them. These user role assignment features are no longer supported by the AuthzForce server itself, but should be delegated to the Identity Management GE.

4.7.2 Performance Tuning

For Tomcat and JVM tuning, we strongly recommend reading and applying - when relevant - the guidelines from the following links:

- Performance tuning best practices for VMware Apache Tomcat;
- Tuning Tomcat Performance For Optimum Speed;
- How to optimize tomcat performance in production;
- Apache Tomcat Tuning Guide for REST/HTTP APIs.

Last but not least, consider tuning the OS, hardware (CPU, RAM. . .), network, using load-balancing, high-availability solutions, and so on.

CHAPTER 5

User and Programmers Guide

This guide explains how to use the API to manage XACML-based access control policies and provide authorization decisions based on such policies and the context of a given access request.

If you have been using a previous version of AuthzForce, check the release notes **to know what is changed and what is new.**

5.1 Background and Detail

This User and Programmers Guide applies to the reference implementation of the Authorization PDP GE which is part of FIWARE Security Architecture. Please find more information about this Generic Enabler in the following Open Specification.

5.2 User Guide

Since the Authorization PDP is a Generic Enabler which provides backend functionality to other applications (e.g. Generic Enablers or end user facing applications) and security administrators, we do not distinguish between the User and Programmers Guide. Please refer to the Programmers Guide section for more information.

5.3 Programmer Guide

AuthzForce provides the following APIs:

- PDP API (PDP = Policy Decision Point in the XACML terminology): provides an API for getting authorization decisions computed by a XACML-compliant access control engine;
- PAP API (PAP = Policy Administration Point in XACML terminology): provides API for managing XACML policies to be handled by the Authorization Service PDP.

The full API (RESTful) is described by a document written in the Web Application Description Language format (WADL) and associated XML schema files available in Authzforce rest-api-model project files.

XACML is the main international OASIS standard for access control language and request-response formats, that addresses most use cases of access control. AuthzForce supports the full core XACML 3.0 language; therefore it allows to enforce generic and complex access control policies.

5.3.1 General recommendations for developers

In all the sample XML outputs shown in the next sections, the XML namespace prefix of any API response element, such as the XACML `Response` element, might vary from an AuthzForce run time to another, but it is always the same XML element as the prefix is always mapped to the same namespace, such as `urn:oasis:names:tc:xacml:3.0:core:schema:wd-17` (XACML 3.0 namespace) for the XACML `Response`. Therefore, any valid (namespace-aware) XML parser will handle it equally, no matter the namespace prefix. Beware of that XML namespace-prefix mapping issue if you are parsing XML manually.

We strongly recommend developers to use XML-schema-aware software with XML schema validation enabled for all XML processing. This will make troubleshooting easier, and save you a lot of trouble. You can find the XML schemas and an example of Java client code with schema validation in the Authzforce rest-api-model project files.

5.3.2 Attribute-Based Access Control

AuthzForce provides Attribute-Based Access Control. To understand what is meant by *attribute* in the context of access control, below is the list of standard categories of attributes identified by the XACML standard:

- Subject attributes: the subject is an actor (human, program, device, etc.) requesting access to a resource; attributes may be user ID, Organization, Role, Clearance, etc. In fact, XACML enables you to be more specific about the type of subject, e.g. intermediary subject, requesting machine, etc.
- Resource attributes: the resource is a passive entity (from the access control perspective) on which subject requests to act upon (e.g. data but also human, device, application, etc.); resource attributes may be resource ID, URL, classification, etc.
- Action attributes: the action is the action that the subject requests to perform on the resource (e.g. create, read, delete); attributes may be action ID, parameter A, parameter B, etc.
- Environment attributes: anything else, e.g. current time, CPU load of the PEP/PDP, global threat level, etc.

If this is not enough, XACML enables you to make your own attribute categories as well.

5.3.3 Product info API

You can get product information (name, version...) with following HTTP request:

- Method: GET
- Path: `/version`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

5.3.4 Domain Management API

The API allows AuthzForce application administrators or administration interfaces to create domains for the users, and remove domains once they are no longer used. This part of the API is described in the section *Policy Domain Administration*.

The API is provided over HTTP in order to comply with the test assertions `urn:oasis:names:tc:xacml:3.0:profile:rest:assertion:http:client` and `urn:oasis:names:tc:xacml:3.0:profile:rest:assertion:http:server` of REST Profile of XACML v3.0 Version 1.0.

Each AuthzForce domain represents an independent *RESTful XACML system*, in the context of the REST Profile of XACML v3.0 Version 1.0.

End-users may retrieve the domain's resource content as follows:

- Method: GET
- Path: `/domains/{domainId}`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example, this request gets the resource for domain `iMnxv7sDEeWFwqVFFMDLTQ`

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

If the domain exists, the response goes:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<domain
 xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
 xmlns:atom="http://www.w3.org/2005/Atom">
 <properties externalId="test-domain1">
  <description>Test domain</description>
 </properties>
 <childResources>
  <atom:link rel="item" href="/properties" title="Domain properties"/>
  <atom:link rel="item" href="/pap" title="Policy Administration Point"/>
  <atom:link
   rel="http://docs.oasis-open.org/ns/xacml/relation/pdp"
   href="/pdp" title="Policy Decision Point"/>
 </childResources>
</domain>
```

If a domain with such ID does not exist, an error 404 is returned.

Therefore, in the context of the REST Profile of XACML v3.0 Version 1.0, the location of the single entry point of a domain-specific RESTful XACML system is `/domains/{domainId}`, and you may get the link to the PDP from the response for the `GET` request to this entry point location, looking for the link relation `http://docs.oasis-open.org/ns/xacml/relation/pdp`. In this respect, we comply with test assertions `urn:oasis:names:tc:xacml:3.0:profile:rest:assertion:home:documentation`, `urn:oasis:names:tc:xacml:3.0:profile:rest:assertion:home:status` and `urn:oasis:names:tc:xacml:3.0:profile:rest:assertion:home:pdp` of the REST Profile of XACML v3.0 Version 1.0.

The API also allows users to update certain properties of the domain allocated to them:

- An **externalId** (optional) for the domain, which users/clients can modify and more easily use as reference, as opposed to the unique and read-only domain ID assigned by the API - once and for all - when the domain is

created;

- A **description** of the domain (optional).

You may retrieve the current domain properties as follows:

- Method: GET
- Path: `/domains/{domainId}/properties`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example, this request gets the properties of domain `iMnxv7sDEeWFwqVFFMDLTQ`. In this case, there is no specific property, which is the case by default:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/properties
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

The response goes:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<domainProperties
   xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
   externalId="test-domain1">
   <description>Test domain</description>
</domainProperties>
```

You may update the domain properties as follows:

- Method: PUT
- Path: `/domains/{domainId}/properties`
- Headers:
  - Content-Type: `application/xml; charset=UTF-8`
  - Accept: `application/xml; charset=UTF-8`
- Body: new properties.

For example, this request sets the `externalId` property to `my-domain-123`:

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/properties
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<domainProperties
 xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
 externalId="my-domain-123" />
```

The response is the new properties.

As a result, the domain's external ID `my-domain-123` points to the domain `iMnxv7sDEeWFwqVFFMDLTQ`. Clients may only rely on the externalId under their control to recover the API-defined domain ID, before they begin to use other API operations that require the API-defined domain ID. Indeed, clients may look up the API-defined ID corresponding to a given `externalId` as follows:

```
GET /domains?externalId=my-domain-123
HTTP/1.1
Accept: application/xml; charset=UTF-8
```

The response gives the corresponding domain ID in a link `href` attribute:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resources
  xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
  xmlns:atom="http://www.w3.org/2005/Atom">
  <atom:link rel="item" href="iMnxv7sDEeWFwqVFFMDLTQ" title="iMnxv7sDEeWFwqVFFMDLTQ"/>
</resources>
```

5.3.5 Policy Administration API

The PAP is used by policy administrators to manage the policy repository from which the PDP loads the enforced policies. The PAP supports multi-tenancy in the form of generic administration domains that are separate from each other. Each policy administrator (except the Superadmin) is in fact a domain administrator, insofar as he is allowed to manage the policy for one or more specific domains. Domains are typically used to support isolation of tenants (one domain per tenant).

5.3.5.1 Adding and updating Policies

The PAP provides a RESTful API for adding and updating policies to a specific domain. HTTP requests to this API must be formatted as follows:

- Method: POST
- Path: `/domains/{domainId}/pap/policies`
- Headers:
  - Content-Type: `application/xml; charset=UTF-8`
  - Accept: `application/xml; charset=UTF-8`
- Body: XACML PolicySet as defined in the XACML 3.0 schema.

Example of request given below:

```
POST /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet
 xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 PolicySetId="P1"
 Version="1.0"
 PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
 <Description>Sample PolicySet</Description>
 <Target />
 <Policy
  PolicyId="MissionManagementApp"
  Version="1.0"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
  <Description>Policy for MissionManagementApp</Description>
  <Target>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">MissionManagementApp</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
```

(continues on next page)

(continued from previous page)

```
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string"
      MustBePresent="true" />
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Rule RuleId="MissionManager_role_can_manage_team" Effect="Permit">
  <Description>Only MissionManager role authorized to manage the mission team</Description>
  <Target>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">Team</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
       AttributeId="urn:thales:xacml:2.0:resource:sub-resource-id"
       DataType="http://www.w3.org/2001/XMLSchema#string"
       MustBePresent="true" />
     </Match>
    </AllOf>
   </AnyOf>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">manage</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
       DataType="http://www.w3.org/2001/XMLSchema#string"
       MustBePresent="true" />
     </Match>
    </AllOf>
   </AnyOf>
  </Target>
  <Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
    <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
     <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">MissionManager</AttributeValue>
     <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
      DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"
      Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" />
   </Apply>
  </Condition>
 </Rule>
</Policy>
</PolicySet>
```

The HTTP response status is 200 with a link to manage the new policy, if the request was successful. The link is made of the policy ID and version separated by '/'.

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<atom:link xmlns:atom="http://www.w3.org/2005/Atom"
  rel="item" href="P1/1.0" title="Policy 'P1' v1.0"/>
```

To update a policy, you add a new version of the policy, i.e. you send the same request as above, but with a higher `Version` value.

WARNING: after you add/update a policy, it is not necessarily used, i.e.evaluated, by the PDP. The PDP starts the evaluation with the root policy specified in the *Policy Decision (PDP) Properties*. Therefore, only this root policy and any other one referenced (directly or indirectly) by this root policy is applicable. See the section *Policy Decision (PDP) Properties* to learn how to verify applicable policies and change the root policy.

WARNING: Although AuthzForce Server supports `application/json` media type as well for sending/getting policies in JSON format, it is still experimental for various reasons. One of which is a strong limitation that you should be much aware before using it for complex XACML policies: XML schema type definitions using a repeated''choice'' (between different element types) or a polymorphic sequence with `maxOccurs > 1` are not handled properly in JSON, at least not in any standard way or without significant customization of JSON processing. For example of such polymorphic sequence, an XACML Apply element may contain multiple Expression elements in a sequence, and an Expression may be an Apply again, or an AttributeValue, or an AttributeDesignator, or a VariableReference, etc. For example of repeated `choice`, a XACML PolicySet may contain P1, then PS1, then P2, then PS2, where `P` stands for XACML Policy and `PS` for XACML PolicySet. With the well-known simple conventions like the one used by AuthzForce Server for XML-to-JSON mapping, this is mapped to two separate JSON arrays, one for Policy element(s) (`[P1, P2]`) and one for PolicySet element(s) (`[PS1, PS2]`). Therefore, the originally intended evaluation order is lost! It gets even worse if you use PolicySetIdReference element(s) as well (PolicyIdReference is out of the question since not supported by the API). Of course, there are solutions such as adding a wrapping JSON object with a key called `PolicyOrPolicySetOrPolicySetIdReference` with an array as value where each item must have a type information to inform the consumer whether it is a Policy, PolicySet or PolicySetIdReference. This kind of solution is used in JAXB for instance to map XML to Java model (except the array is replaced by a Java collection). Like in JAXB to Java, this introduces some extra complexity to JSON processing that makes the JSON alternative lose much of its appeal compared to XML. In short, **you should not use JSON for policies either mixing XACML Policy, PolicySet or PolicySetIdReference elements within the same PolicySet; or Expressions within the same Apply**.

5.3.5.2 Getting Policies and Policy Versions

Once added to the domain as shown previously, you can get the policy by its ID as follows:

- Method: GET
- Path: `/domains/{domainId}/pap/policies/{policyId}`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1 HTTP/1.1 Accept: application/xml; charset=UTF-8
```

The response is the list of links to the versions of the policy `P1` available in the domain `iMnxv7sDEeWFwqVFFMDLTQ`:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resources
  xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
  xmlns:atom="http://www.w3.org/2005/Atom">
    <atom:link rel="item" href="1.0"/>
    <atom:link rel="item" href="1.1"/>
    <atom:link rel="item" href="2.0"/>
    <atom:link rel="item" href="2.1"/>
    <atom:link rel="item" href="2.2"/>
    ...
</resources>
```

As the `href` values are telling you, you may get a specific version of the policy as follows:

- Method: GET
- Path: `/domains/{domainId}/pap/policies/{policyId}/{version}`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1/1.0 HTTP/1.1 Accept: application/xml; charset=UTF-8
```

The response is the policy document (XACML PolicySet) in this version.

You may use the special keyword `latest` as version here to get the latest version of a given policy; e.g. URL path `/domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1/latest` points to the latest version of the policy `P1` in domain `iMnxv7sDEeWFwqVFFMDLTQ`.

Last but not least, you may get all policies in the domain as follows:

- Method: GET
- Path: `/domains/{domainId}/pap/policies`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies
HTTP/1.1
Accept: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resources
  xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
  xmlns:atom="http://www.w3.org/2005/Atom">
    <atom:link rel="item" href="root"/>
    <atom:link rel="item" href="P1"/>
    <atom:link rel="item" href="P2"/>
</resources>
```

5.3.5.3 Removing Policies and Policy Versions

You may remove a policy version from the domain as follows:

- Method: DELETE
- Path: `/domains/{domainId}/pap/policies/{policyId}/{version}`
- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
DELETE /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1/1.0 HTTP/1.1 Accept: application/xml; charset=UTF-8
```

The response is the removed policy document (XACML PolicySet) in this version.

You may remove a policy, i.e. all versions of a policy from the domain as follows:

- Method: DELETE
- Path: `/domains/{domainId}/pap/policies/{policyId}`

- Headers:
  - Accept: `application/xml; charset=UTF-8`

For example:

```
DELETE /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies/P1 HTTP/1.1 Accept: application/xml; charset=UTF-8
```

The response is the list of links to all the removed versions of the policy, similar to the the GET request on the same URL.

5.3.5.4 Re-usable Policies (e.g. for Hierarchical RBAC)

The PAP API supports policies that have references to other policies existing in the domain. This allows to include/reuse a given policy from multiple policies, or multiple parts of the same policy, by means of XACML <PolicySetIdReference> elements. One major application of this is Hierarchical RBAC. You can refer to the XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile specification for how to achieve hierarchical RBAC with <PolicySetIdReference> elements.

For example, I want to define a role *Employee* and a role *Manager* derived from *Employee*. In other words, permissions of an *Employee* are included in the permissions of a *Manager*. In order to create this role hierarchy, we first add the Employee's *Permission PolicySet*:

```
POST /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<PolicySet
 xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 PolicySetId="PPS:Employee"
 Version="1.0"
 PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
 <Description>Permissions specific to the Employee role</Description>
 <Target />
 <Policy
  PolicyId="PP:Employee"
  Version="1.0"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
  <Target />
  <Rule RuleId="Permission_to_create_issue_ticket" Effect="Permit">
   <Target>
    <AnyOf>
     <AllOf>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">https://acme.com/tickets</AttributeValue>
       <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
        DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
      </Match>
     </AllOf>
    </AnyOf>
    <AnyOf>
     <AllOf>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">POST</AttributeValue>
       <AttributeDesignator
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string"
        MustBePresent="true" />
```

(continues on next page)

(continued from previous page)

```
      </Match>
     </AllOf>
    </AnyOf>
   </Target>
  </Rule>
 </Policy>
</PolicySet>
```

Then we add the role-based hierarchical policy defining the Employee role and the Manager role, both with a reference (<PolicySetIdReference>) to the Employee's *Permission PolicySet* added previously. The Manager role has one policy more, so more permissions:

```
POST /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/policies
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicySetId="rbac:policyset"
 Version="1.0"
 PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
 <Description>Root PolicySet</Description>
 <Target />
 <PolicySet PolicySetId="RPS:Employee" Version="1.0"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
  <Description>Employee Role PolicySet</Description>
  <Target>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
       DataType="http://www.w3.org/2001/XMLSchema#string">Employee</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
       AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
       DataType="http://www.w3.org/2001/XMLSchema#string"
       MustBePresent="true" />
     </Match>
    </AllOf>
   </AnyOf>
  </Target>
  <PolicySetIdReference>PPS:Employee</PolicySetIdReference>
 </PolicySet>
 <PolicySet PolicySetId="RPS:Manager" Version="1.0"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
  <Description>Manager Role PolicySet</Description>
  <Target>
   <AnyOf>
    <AllOf>
     <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Manager</AttributeValue>
      <AttributeDesignator
       Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
       AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
       DataType="http://www.w3.org/2001/XMLSchema#string"
       MustBePresent="true" />
     </Match>
    </AllOf>
   </AnyOf>
  </Target>
  <Policy PolicyId="PP1:Manager" Version="1.0"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
   <Description>Permissions specific to Manager Role</Description>
```

(continues on next page)

(continued from previous page)

```
   <Target />
   <Rule
    RuleId="Permission_to_create_new_project" Effect="Permit">
    <Target>
     <AnyOf>
      <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
         DataType="http://www.w3.org/2001/XMLSchema#string">https://acme.com/projects</AttributeValue>
        <AttributeDesignator
         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
         AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
         DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true" />
       </Match>
      </AllOf>
     </AnyOf>
     <AnyOf>
      <AllOf>
       <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">POST</AttributeValue>
        <AttributeDesignator
         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
         DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
       </Match>
      </AllOf>
     </AnyOf>
    </Target>
   </Rule>
  </Policy>
  <!-- This role is senior to the Employee role, therefore includes the Employee role Permission
   PolicySet -->
  <PolicySetIdReference>PPS:Employee</PolicySetIdReference>
 </PolicySet>
</PolicySet>
```

You may add more policies for more roles as you wish. Once you are satisfied with your role hierarchy, you may apply your new RBAC policy by updating the domain's root policy reference (this may not be necessary if you reused the same root policy ID as before, in which case your policy is already active by now):

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdpPropertiesUpdate xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
 <rootPolicyRefExpression>rbac:policyset</rootPolicyRefExpression>
</pdpPropertiesUpdate>
```

The policy is now enforced by the PDP as described in the next section.

5.3.5.5 Policy Repository (PRP) Properties

Administrators (global or domain-specific) may configure the policy repository with the following properties:

- `maxPolicyCount`: optional, stricly positive integer that indicates the maximum number of policies on a domain, no limit if undefined.
- `maxVersionCountPerPolicy`: optional, stricly positive integer that indicates the maximum number of versions per policy, no limit if undefined.

- `versionRollingEnabled`: boolean, true if and only if policy versions should be rolled over, i.e. when `maxVersionCountPerPolicy` has been reached, oldest versions are automatically removed to make place.

For example, below is a HTTP GET request and response for the policy repository properties of domain `iMnxv7sDEeWFwqVFFMDLTQ`:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/prp.properties
Accept: application/xml

-

HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<prpProperties xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
   <maxPolicyCount>10</maxPolicyCount>
   <maxVersionCountPerPolicy>10</maxVersionCountPerPolicy>
   <versionRollingEnabled>true</versionRollingEnabled>
</prpProperties>
```

The HTTP PUT request to update the properties has a body that is similar to the GET response:

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/prp.properties
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<prpProperties xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
   <maxPolicyCount>4</maxPolicyCount>
   <maxVersionCountPerPolicy>2</maxVersionCountPerPolicy>
   <versionRollingEnabled>true</versionRollingEnabled>
</prpProperties>
```

The response format is the same as for the GET request.

5.3.5.6 Policy Decision (PDP) Properties

Administrators (global or domain-specific) may configure the PDP engine with the following properties:

- `rootPolicyRefExpression`: reference - in the form of a XACML PolicySetIdReference - to the root policy. The root policy is the policy from which the PDP starts the evaluation. A policy matching this reference must exist on the domain, therefore it must have been added in the way described in *Adding and updating Policies*. If there is no specific `Version` in the reference, the latest matching policy version is selected.
- `feature` elements: enable particular PDP features. Each `feature` has an ID, `type` and `enabled` flag saying whether the feature is enabled or not.

Supported PDP features (IDs) by `type`:

- Type `urn:ow2:authzforce:feature-type:pdp:core`: PDP core engine features (as opposed to other types related to PDP extensions).
  - `urn:ow2:authzforce:feature:pdp:core:strict-attribute-issuer-match`: strict matching of attribute `Issuer` values in XACML Requests against corresponding attribute designators' `Issuer` values in policies. This means that an <`AttributeDesignator`> without `Issuer` only matches request Attributes without `Issuer` (and same AttributeId, Category...). This mode is not fully compliant with XACML 3.0 Core specifcation of AttributeDesignator (§5.29), in the case that the Issuer is indeed not present on a AttributeDesignator, but it may perform better and is recommended when all AttributeDesignators have an Issuer. Reminder: XACML 3.0 Core specifcation of AttributeDesignator (§5.29) says: *If the Issuer is not present in the attribute designator, then the matching of the attribute to the named attribute SHALL be governed by AttributeId and DataType attributes alone.*

  - `urn:ow2:authzforce:feature:pdp:core:xpath-eval`: enables support for XACML AttributeSelectors and datatype `urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression`. If this feature is disabled, only standard XACML 3.0 Core datatypes marked *M*, i.e. mandatory, are supported. Since `xpathExpression` is optional in the standard, it is therefore not supported unless this feature is enabled. **This feature is experimental and may have a negative impact on performance. Use with caution.**

- Type `urn:ow2:authzforce:feature-type:pdp:request-preproc`: XACML (Individual) Request preprocessor (*Individual* means that even if the XACML Multiple Decision Profile is active, the request preprocessor applies to each *Individual* Decision Request as defined in the Profile). As a convention, request preprocessor IDs with suffix `-lax` allow multivalued attributes in form of duplicate Attribute elements (with same meta-data) in the same Attributes element of a Request, in order to accept multivalued attributes in conformance with XACML 3.0 Core specification of Multivalued attributes (§7.3.3). Request preprocessor IDs with suffix `-strict` do not allow this behavior, i.e. multivalued attributes must be formed by grouping all AttributeValue elements in the same Attribute element (instead of duplicate Attribute elements), therefore they do not fully comply with XACML 3.0 Core specification of Multivalued attributes (§7.3.3). However, they perform usually better than their `-lax` counterparts since it simplifies the Request and allows parsing optimizations by the PDP. Below is an example of Request that would not be accepted by a `-strict` request preprocessor because of duplicate Attribute:

```
<Request
 xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" ReturnPolicyIdList="false" CombinedDecision=
↪"false"> <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role" IncludeInResult="false">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">CS0</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role" IncludeInResult="false">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">CT0</AttributeValue>
    </Attribute> ...
 </Attributes> ...
</Request>
```

Below is the equivalent of the previous Request in a form that is accepted by a `-strict` request preprocessor (no duplicate Attribute):

```
<Request
 xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" ReturnPolicyIdList="false" CombinedDecision=
↪"false"> <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role" IncludeInResult="false">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">CS0</AttributeValue>
↪<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">CT0</AttributeValue>
    </Attribute> ...
 </Attributes> ...
</Request>
```

Available request preprocessor IDs:

  - *urn:ow2:authzforce:feature:pdp:request-preproc:default-lax* and *urn:ow2:authzforce:feature:pdp:request-preproc:default-strict*: supports only XACML Request elements marked as *mandatory* in XACML 3.0 Core specification (§10.2.1) (in particular, **no** support for Multiple Decision Profile);

  - *urn:ow2:authzforce:feature:pdp:request-preproc:multiple:repeated-attribute-categories-lax* and *urn:ow2:authzforce:feature:pdp:request-preproc:multiple:repeated-attribute-categories-strict*: Provides the functionality identified by *urn:oasis:names:tc:xacml:3.0:profile:multiple:repeated-attribute-categories* in XACML v3.0 Multiple Decision Profile Version 1.0 (§3.3)

  Only one request preprocessor may be enabled at at time.

- Types `urn:ow2:authzforce:feature-type:pdp:data-type` and `urn:ow2:authzforce:feature-type:pdp:function`: PDP extensions providing *non-core* XACML data types and functions respectively, i.e. not specified in XACML 3.0 Core standard §10.2.7 and §10.2.8 respectively. More information in next section *PDP Extensions*.

Follow the example of request/response below to get the current PDP properties in domain `iMnxv7sDEeWFwqVFFMDLTQ`:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
Accept: application/xml

-

HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdpProperties
 xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
 lastModifiedTime="2016-05-28T14:21:35.730Z">
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:core"
  enabled="false">urn:ow2:authzforce:feature:pdp:core:strict-attribute-issuer-match</feature>
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:request-preproc"
  enabled="true">urn:ow2:authzforce:feature:pdp:request-preproc:default-lax</feature>
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:request-preproc"
  enabled="false">urn:ow2:authzforce:feature:pdp:request-preproc:default-strict</feature>
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:request-preproc"
  enabled="false">urn:ow2:authzforce:feature:pdp:request-preproc:multiple:repeated-attribute-categories-strict
↪</feature>
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:request-preproc"
  enabled="false">urn:ow2:authzforce:feature:pdp:request-preproc:multiple:repeated-attribute-categories-lax</
↪feature>
 ...(content omitted)...
 <rootPolicyRefExpression>root</rootPolicyRefExpression>
 <applicablePolicies>
  <rootPolicyRef Version="0.1.0">root</rootPolicyRef>
  <refPolicyRef Version="1.0">PPS:Employee</refPolicyRef>
  <refPolicyRef Version="1.0">PPS:Manager</refPolicyRef>
  ...(content omitted)...
 </applicablePolicies>
</pdpProperties>
```

As you can see, the GET response provides extra information such as:

- `lastModifiedTime`: the last time the PDP was reloaded (due to a change of root policy for instance);
- `applicablePolicies`: the actual root policy (`rootPolicyRef` element) version selected for evaluation according to the `rootPolicyRefExpression`, and any policy referenced from it (`refPolicyRef` elements) directly or indirectly via `PolicySetIdReference`.

The HTTP PUT request to update the PDP properties goes as follows:

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdpPropertiesUpdate xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:request-preproc"
  enabled="true">urn:ow2:authzforce:feature:pdp:request-preproc:multiple:repeated-attribute-categories-lax</
↪feature>
 <rootPolicyRefExpression>root</rootPolicyRefExpression>
</pdpPropertiesUpdate>
```

This example sets the root policy reference to the latest version of the policy with `PolicySetId = 'root'` that must exist in the domain (see *Adding and updating Policies*), and enables support for the XACML Multiple Decision pro-

file with repeated attribute categories (*urn:oasis:names:tc:xacml:3.0:profile:multiple:repeated-attribute-categories*). Notice that only one feature element in the request although it is not the only one PDP feature. In this case, the API assumes that all features missing from the request must be disabled. Therefore, it is only necessary to send the **enabled** features in the request.

5.3.5.7 PDP Extensions

Non-core (not defined in XACML 3.0 Core standard) PDP behavior and features may be implemented by various types of extensions, particularly to support specific XACML Profiles:

- Attribute Datatypes: to support extra XACML datatypes, e.g. from DLP/NAC Profile;
- Functions: to support extra XACML functions, e.g. from DLP/NAC Profile;
- Attribute Providers: to customize the way attribute value are retrieved outside the PEP's Request.
- Request Preprocessors: to customize the processing of individual decision requests;
- Result Postprocessors: to customize the processing of XACML Results in Response;
- Combining algorithms: to add custom policy or rule combining algorithms.

5.3.5.7.1 Attribute Datatype extensions

The XACML 3.0 Core standard allows to use extra attribute data types not defined in the standard. Before you can use such datatypes in Authzforce API, you must implement and provide it as an Attribute Datatype extension, or get it from a third party as such; and then you deploy it on Authzforce server and enable it on a specific domain. The AuthzForce project also provides a separate Datatype extension example for documentation and testing purposes. If you wish to make your own Attribute Datatype extension, read on the next section. If you wish to test the example provided by AuthzForce or if you have another one ready for use, you may jump to the section *Integrating an Attribute Datatype extension into AuthzForce Server*.

5.3.5.7.1.1 Making an Attribute Datatype extension

The steps to make your own Attribute Datatype extension for AuthzForce go as follows:

1. Create a Maven project with `jar` packaging type and following Maven dependency:

```
<dependency>
  <groupId>org.ow2.authzforce</groupId>
  <artifactId>authzforce-ce-core-pdp-api</artifactId>
  <version>18.0.1</version>
  <scope>provided<scope>
</dependency>
```

2. Create your attribute datatype factory and value instance class (as in the *Factory* design pattern). The factory class must be public, and implement interface `org.ow2.authzforce.core.pdp.api.value.DatatypeFactory<AV>`, where `AV` stands for your *AttributeValue Implementation Class*, i.e. the concrete attribute value implementation class; and the factory class must have a public no-argument constructor or no constructor.

   To facilitate the implementation process, instead of implementing this `DatatypeFactory` interface directly, you should extend one of the following `DatatypeFactory` sub-classes when it applies:

   - `org.ow2.authzforce.core.pdp.api.value.SimpleValue.StringContentOnlyFactory<AV>`: to be extended for implementing text-only primitive datatypes (equivalent to simple XML types). You may use AuthzForce TestDNSNameWithPortValue class (used for AuthzForce unit tests) as an example.

This example provides a test implementation of datatype `dnsName-value` defined in XACML Data Loss Prevention / Network Access Control (DLP/NAC) Profile Version 1.0. In this example, the static nested class `Factory` is the one extending `org.ow2.authzforce.core.pdp.api.value.SimpleValue.StringContentOnlyFactory<TestDNSNameWithPortValue>`. Such a class has a factory method (`TestDNSNameWithPortValue getInstance(String val)`) that takes a string argument corresponding to the text in the XACML AttributeValue (which must not contain any XML element or attribute).

- `org.ow2.authzforce.core.pdp.api.value.SimpleValue.Factory<AV>`: to be extended for implementing primitive XACML datatypes with XML attributes (equivalent to complex XML types with simple content). An example of such datatype is `xpathExpression` which requires an XML attribute named `XPathCategory`. Note that the datatype `xpathExpression` is natively supported but enabled only if feature `urn:ow2:authzforce:feature:pdp:core:xpath-eval` is enabled in the PDP configuration.
- `org.ow2.authzforce.core.pdp.api.value.BaseDatatypeFactory<AV>`: to be extended for implementing structured attributes (XACML 3.0 Core, §8.2) (equivalent to complex XML types with complex content). You may use AuthzForce TestXACMLPolicyAttributeValue class (used for AuthzForce unit tests) as an example. In this example, the static nested class `Factory` is the one extending `org.ow2.authzforce.core.pdp.api.value.BaseDatatypeFactory<TestXACMLPolicyAttributeValue>`. Such a class has a factory method `TestXACMLPolicyAttributeValue getInstance(List<Serializable> content, Map<QName, String> otherAttributes, ...)` that creates an instance of your *AttributeValue Implementation Class*, i.e. `TestXACMLPolicyAttributeValue` in this case. where the argument `otherAttributes` represents the XML attributes and argument `content` the mixed content of a XACML AttributeValue parsed by JAXB.

3. When your implementation class is ready, create a text file `org.ow2.authzforce.core.pdp.api.PdpExtension` in folder `src/main/resources/META-INF/services` (you have to create the folder first) and put the fully qualified name of your implementation class on the first line of this file, like in the example from AuthzForce source code.
4. Run Maven `package` to produce a JAR from the Maven project.

Now you have an Attribute Datatype extension ready for integration into AuthzForce Server, as explained in the next section.

5.3.5.7.1.2 Integrating an Attribute Datatype extension into AuthzForce Server

This section assumes you have an Attribute Datatype extension in form of a JAR, typically produced by the process described in the previous section. You may use AuthzForce PDP Core Tests JAR if you only wish to test the examples in this documentation. This JAR is available on Maven Central: groupId= `org.ow2.authzforce`, artifactId= `authzforce-ce-core-pdp-testutils`, version= `17.1.0`.

The steps to integrate the extension into the AuthzForce Server go as follows:

1. Make the JAR - and any extra dependency - visible from the AuthzForce webapp in Tomcat. One way to do it consists to copy the JAR (e.g. `authzforce-ce-core-pdp-testutils-16.0.0.jar` in our example) into `/opt/authzforce-ce-server/webapp/WEB-INF/lib`. For other ways, please refer to Tomcat HowTo.
2. Finally, restart Tomcat to apply changes.

5.3.5.7.1.3 Enabling an Attribute Datatype extension on a domain

Once you have deployed the extension on Authzforce, following previous instructions, you are ready to enable it on a specific domain's PDP by updating the PDP properties with an enabled `feature` of type

`urn:ow2:authzforce:feature-type:pdp:data-type` and value equal to the ID returned by the method `getId()` of the extension's factory implementation class. The following example enables the datatype `dnsName-value` (defined in DLP/NAC profile) on the PDP, provided that the AuthzForce PDP Core Tests JAR has been deployed (see previous section):

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdpPropertiesUpdate xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:data-type"
  enabled="true">urn:oasis:names:tc:xacml:3.0:data-type:dnsName-value</feature>
 <rootPolicyRefExpression>root</rootPolicyRefExpression>
</pdpPropertiesUpdate>
```

5.3.5.7.2 Function Extensions

The XACML 3.0 Core standard allows to use extra functions not defined in the standard. Before you can use such functions in Authzforce API, you must implement and provide it as an Function extension, or get it from a third party as such; and then you deploy it on Authzforce server and enable it on a specific domain. The AuthzForce project also provides a separate Function extension example for documentation and testing purposes. If you wish to make your own Function extension, read on the next section. If you wish to test the example provided by AuthzForce or if you have another one ready for use, you may jump to the section *Integrating a Function extension into AuthzForce Server*.

5.3.5.7.2.1 Making a Function extension

The steps to make your own Function extension go as follows:

1. Create a Maven project with `jar` packaging type and following Maven dependency:

```
<dependency>
 <groupId>org.ow2.authzforce</groupId>
 <artifactId>authzforce-ce-core-pdp-api</artifactId>
 <version>18.0.1</version>
 <scope>provided<scope>
</dependency>
```

2. A Function extension class must implement interface `org.ow2.authzforce.core.pdp.api.func.Function`; and have a public no-argument constructor or no constructor. Instead of implementing this `Function` interface directly, you should extend one of the following `Function` sub-classes whenever possible, depending on your needs:

   - `org.ow2.authzforce.core.pdp.api.func.ComparisonFunction`: to be extended for implementing comparison functions `type-greater-than`, `type-greater-than-or-equal`, `type-less-than` and `type-less-than-or-equal`. Examples from XACML 3.0 Core standard: see §A.3.6 and §A.3.8.
   - `org.ow2.authzforce.core.pdp.api.func.EqualTypeMatchFunction`: to be extended for implementing match functions with two parameters of same type‘. Examples from XACML 3.0 Core standard: equality functions in §A.3.1, `x500name-match`, `string-starts-with`. You may use AuthzForce TestDNSNameValueEqualFunction class (used for AuthzForce unit tests) as an example. This example provides a test implementation of function `dnsName-value-equal` defined in XACML Data Loss Prevention / Network Access Control (DLP/NAC) Profile Version 1.0.
   - `org.ow2.authzforce.core.pdp.api.func.NonEqualTypeMatchFunction`: to be extended for implementing match functions with two parameters of different type. Examples from XACML 3.0 Core standard: `rfc822Name-match`, `anyURI-starts-with`, `dnsName-regexp-match`.

- `org.ow2.authzforce.core.pdp.api.func.HigherOrderBagFunction`: to be extended for implementing higher-order bag functions. Examples from XACML 3.0 Core standard are functions in §A.3.12.
- `org.ow2.authzforce.core.pdp.api.func.SingleParameterTypedFirstOrderFunction`: to be extended for implementing first-order functions having all parameters of the same type, when previous cases do not apply. Examples from XACML 3.0 Core standard are logical `and`, `or` or `not` in §A.3.5.
- `org.ow2.authzforce.core.pdp.api.func.MultiParameterTypedFirstOrderFunction`: to be extended for implementing first-order functions having at least two different types of parameters, when previous cases do not apply. Examples from XACML 3.0 Core standard are logical `n-of` and `*-substring` functions.
- `org.ow2.authzforce.core.pdp.api.func.BaseFunction`: to be extended for implementing functions when none of the previous cases apply.

3. When your implementation class is ready, create a text file `org.ow2.authzforce.core.pdp.api.PdpExtension` in folder `src/main/resources/META-INF/services` (you have to create the folder first) and put the fully qualified name of your implementation class on the first line of this file, like in the example from Authzforce source code.
4. Run Maven `package` to produce a JAR from the Maven project.

Now you have a Function extension ready for integration into AuthzForce Server, as explained in the next section.

5.3.5.7.2.2 Integrating a Function extension into AuthzForce Server

This section assumes you have a Function extension in form of a JAR, typically produced by the process described in the previous section. You may use AuthzForce PDP Core Tests JAR if you only wish to test the examples in this documentation. This JAR is available on Maven Central: groupId= `org.ow2.authzforce`, artifactId= `authzforce-ce-core-pdp-testutils`, version= `17.1.0`.

The steps to integrate the extension into the AuthzForce Server go as follows:

1. Make the JAR - and any extra dependency - visible from the AuthzForce webapp in Tomcat. One way to do it consists to copy the JAR (e.g. `authzforce-ce-core-pdp-testutils-16.0.0.jar` in our example) into `/opt/authzforce-ce-server/webapp/WEB-INF/lib`. For other ways, please refer to Tomcat HowTo.
2. Finally, restart Tomcat to apply changes.

5.3.5.7.2.3 Enabling a Function extension on a domain

Once you have deployed the extension on Authzforce, following previous instructions, you are ready to enable it on a specific domain's PDP by updating the PDP properties with an enabled `feature` of type `urn:ow2:authzforce:feature-type:pdp:function-set` if the extension extends `BaseFunctionSet` class or implements directly its superinterface `FunctionSet`; else use the feature type `urn:ow2:authzforce:feature-type:pdp:function`, and value equal to the ID returned by the method `getId()` of the extension implementation class. The following example enables the function `dnsName-value-equal` and required datatype `dnsName-value` (defined in DLP/NAC profile) on the PDP, provided that the AuthzForce PDP Core Tests JAR has been deployed (see previous section):

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdpPropertiesUpdate xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:data-type"
```

(continues on next page)

(continued from previous page)

```
    enabled="true">urn:oasis:names:tc:xacml:3.0:data-type:dnsName-value</feature>
  <feature
    type="urn:ow2:authzforce:feature-type:pdp:data-type"
    enabled="true">urn:oasis:names:tc:xacml:3.0:data-type:dnsName-value-equal</feature>
  <rootPolicyRefExpression>root</rootPolicyRefExpression>
</pdpPropertiesUpdate>
```

5.3.5.7.3 Combining Algorithm Extensions

The XACML 3.0 Core standard allows to use extra policy/rule combining algorithms not defined in the standard. Before you can use such algorithms in Authzforce API, you must implement and provide it as an Combining Algorithm extension, or get it from a third party as such; and then you deploy it on Authzforce server and enable it on a specific domain. The AuthzForce project also provides a separate Combining Algorithm extension example for documentation and testing purposes. If you wish to make your own Combining Algorithm extension, read on the next section. If you wish to test the example provided by AuthzForce or if you have another one ready for use, you may jump to the section *Integrating a Combining Algorithm extension into AuthzForce Server*.

5.3.5.7.3.1 Making a Combining Algorithm extension

The steps to make your own Combining Algorithm extension go as follows:

1. Create a Maven project with `jar` packaging type and following Maven dependency:

```
<dependency>
  <groupId>org.ow2.authzforce</groupId>
  <artifactId>authzforce-ce-core-pdp-api</artifactId>
  <version>18.0.1</version>
</dependency>
```

2. Create the Java implementation class, either extending class *org.ow2.authzforce.core.pdp.api.combining.BaseCombiningAlg<D>* or, as second resort, implementing interface *org.ow2.authzforce.core.pdp.api.combining.CombiningAlg<D>*, where the type parameter `D` represents the type of elements combined by the algorithm implementation (policy or rule), more precisely `D` must be one of the following:

   - `org.ow2.authzforce.core.pdp.api.Decidable` (recommended option) for a policy/rule combining algorithm implementation, i.e. combining policies and rules equally. For example, although the XACML standard specifies two distinct identifiers for the policy combining version and rule combining version of the *deny-unless-permit* algorithm, the normative algorithm specification in pseudo-code is the same, and is actually implemented by one single Java class in AuthzForce. We strongly recommend this type parameter for your implementation as it makes it more generic and maximizes its reuse.
   - `org.ow2.authzforce.core.pdp.api.policy.PolicyEvaluator` for a policy-only combining algorithm, e.g. the XACML Core standard *only-one-applicable* algorithm, or the *on-permit-apply-second* policy combining algorithm from XACML 3.0 Additional Combining Algorithms Profile Version 1.0. You may use AuthzForce TestOnPermitApplySecondCombiningAlg class (used for AuthzForce unit tests) as an example of implementation for this algorithm.

   This class must have a public no-argument constructor or no constructor.

3. When your implementation class is ready, create a text file `org.ow2.authzforce.core.pdp.api.PdpExtension` in folder `src/main/resources/META-INF/services` (you have to create the folder first) and put the fully qualified name of your implementation class on the first line of this file, like in the example from Authzforce source code.

4. Run Maven `package` to produce a JAR from the Maven project.

Now you have a Combining Algorithm extension ready for integration into AuthzForce Server, as explained in the next section.

5.3.5.7.3.2 Integrating a Combining Algorithm extension into AuthzForce Server

This section assumes you have a Combining Algorithm extension in form of a JAR, typically produced by the process described in the previous section. You may use AuthzForce PDP Core Tests JAR if you only wish to test the examples in this documentation. This JAR is available on Maven Central: groupId= `org.ow2.authzforce`, artifactId= `authzforce-ce-core-pdp-testutils`, version= `17.1.0`.

The steps to integrate the extension into the AuthzForce Server go as follows:

1. Make the JAR - and any extra dependency - visible from the AuthzForce webapp in Tomcat. One way to do it consists to copy the JAR (e.g. `authzforce-ce-core-pdp-testutils-16.0.0.jar` in our example) into `/opt/authzforce-ce-server/webapp/WEB-INF/lib`. For other ways, please refer to Tomcat HowTo.
2. Finally, restart Tomcat to apply changes.

5.3.5.7.3.3 Enabling a Combining Algorithm extension on a domain

Once you have deployed the extension on Authzforce, following previous instructions, you are ready to enable it on a specific domain's PDP by updating the PDP properties with an enabled `feature` of type `urn:ow2:authzforce:feature-type:pdp:combining-algorithm`. The following example enables the combining algorithm `on-permit-apply-second` on the PDP, provided that the AuthzForce PDP Core Tests JAR has been deployed (see previous section):

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdpPropertiesUpdate xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:combining-algorithm"
  enabled="true">urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:on-permit-apply-second</feature>
 <rootPolicyRefExpression>root</rootPolicyRefExpression>
</pdpPropertiesUpdate>
```

5.3.5.7.4 Request Preprocessor Extensions

With AuthzForce *Request Preprocessor* extensions, you can support new ways of processing XACML Requests before evaluation by the PDP engine (e.g. used for implementing XACML v3.0 Multiple Decision Profile Version 1.0 - Repeated attribute categories (§3.3)), for example support alternative request formats/syntax that can be translated to XACML.

Before you can use such extensions in AuthzForce API, you must implement one or get it from a third party as such; and then you deploy it on AuthzForce Server and enable it on a specific domain. Beware that AuthzForce already provides a Request Preprocessor implementing the functionality identified by *urn:oasis:names:tc:xacml:3.0:profile:multiple:repeated-attribute-categories* in XACML v3.0 Multiple Decision Profile Version 1.0 - Repeated attribute categories (§3.3).

If you wish to make your own Request Preprocessor extension, read on the next section. If you wish to test the example provided by AuthzForce or if you have another one ready for use, you may jump to the section *Enabling a Request Preprocessor extension on a domain*.

5.3.5.7.4.1 Making a Request Preprocessor extension

The steps to make your own Request Preprocessor extension for AuthzForce go as follows:

1. Create a Maven project with `jar` packaging type and following Maven dependency:

```
<dependency>
 <groupId>org.ow2.authzforce</groupId>
 <artifactId>authzforce-ce-core-pdp-api</artifactId>
 <version>18.0.1</version>
</dependency>
```

2. Create a Java class implementing interface `org.ow2.authzforce.core.pdp.api.DecisionRequestPreprocessor.Factory`. This class must have a public no-argument constructor or no constructor. This factory class's main goal is to create instances of `org.ow2.authzforce.core.pdp.api.DecisionRequestPreprocessor`. As the latter is an interface, you need a concrete subclass for your implementation. Instead of implementing the interface `DecisionRequestPreprocessor` directly to do so, you should extend class `org.ow2.authzforce.core.pdp.api.io.BaseXacmlJaxbRequestPreprocessor` to facilitate the process whenever possible. You may use AuthzForce SingleDecisionXacmlJaxbRequestPreprocessor.LaxVariantFactory (resp. SingleDecisionXacmlJaxbRequestPreprocessor.StrictVariantFactory) class as an example for *-lax* (resp. *-strict*) request preprocessor. This class implements the minimal XACML 3.0 Core-compliant request preprocessor identified by `urn:ow2:authzforce:feature:pdp:request-preproc:xacml-xml:default-lax` (resp. `urn:ow2:authzforce:feature:pdp:request-preproc:xacml-xml:default-strict`).

3. When your implementation class is ready, create a text file `org.ow2.authzforce.core.pdp.api.PdpExtension` in folder `src/main/resources/META-INF/services` (you have to create the folder first) and put the fully qualified name of your implementation class on the first line of this file, like in the example from AuthzForce source code.

4. Run Maven `package` to produce a JAR from the Maven project.

Now you have a Request Preprocessor extension ready for integration into AuthzForce Server, as explained in the next section.

5.3.5.7.4.2 Integrating a Request Preprocessor extension into AuthzForce Server

This section assumes you have a Request Preprocessor extension in form of a JAR, typically produced by the process described in the previous section. The steps to integrate the extension into the AuthzForce Server go as follows:

1. Make the JAR - and any extra dependency - visible from the AuthzForce webapp in Tomcat. One way to do it consists to copy the JAR (e.g. `authzforce-ce-core-pdp-testutils-13.3.1.jar` in our example) into `/opt/authzforce-ce-server/webapp/WEB-INF/lib`. For other ways, please refer to Tomcat HowTo.

2. Finally, restart Tomcat to apply changes.

5.3.5.7.4.3 Enabling a Request Preprocessor extension on a domain

Once you have deployed the extension on Authzforce, following previous instructions, you are ready to enable it on a specific domain's PDP by updating the PDP properties with an enabled `feature` of type `urn:ow2:authzforce:feature-type:pdp:request-preproc` and value equal to the ID returned by the method `getId()` of the extension's factory implementation class. Please refer to *Policy Decision (PDP) Properties* for examples.

5.3.5.7.5 Result Postprocessor Extensions

With AuthzForce *Result Postprocessors* extensions, you can support new ways of processing XACML Results after evaluation by the PDP engine (e.g. used for implementing XACML v3.0 Multiple Decision Profile Version 1.0 - Requests for a combined decision).

Before you can use such extensions in AuthzForce API, you must implement one or get it from a third party as such. The AuthzForce project also provides a separate Result Postprocessor extension example for documentation and testing purposes. If you wish to make your own Result Postprocessor extension, read on the next section. If you wish to test the example provided by AuthzForce or if you have another one ready for use, you may jump to the section *Enabling a Result Postprocessor extension on a domain*.

5.3.5.7.5.1 Making a Result Postprocessors extension

The steps to make your own Result Postprocessors extension go as follows:

1. Create a Maven project with `jar` packaging type and following Maven dependency:

```
<dependency>
 <groupId>org.ow2.authzforce</groupId>
 <artifactId>authzforce-ce-core-pdp-api</artifactId>
 <version>18.0.1</version>
</dependency>
```

2. Create your ResultPostprocessor factory and concrete implementation class (as in the *Factory* design pattern). The factory class must be public, and implement interface org.ow2.authzforce.core.pdp.api.DecisionResultPostprocessor.Factory. This class must have a public no-argument constructor or no constructor. You may use AuthzForce TestCombinedDecisionXacmlJaxbResultPostprocessor class (used for AuthzForce unit tests) as an example. This example provides a test implementation of feature `urn:oasis:names:tc:xacml:3.0:profile:multiple:combined-decision` from XACML v3.0 Multiple Decision Profile Version 1.0. If you are processing XACML/XML Response, you may extend the more convenient ‘class org.ow2.authzforce.core.pdp.api.io.BaseXacmlJaxbResultPostprocessor.Factoryhttps://www.javadoc.io/doc/org.ow2.authzforce/authzforce-ce-core-pdp-api/latest/org/ow2/authzforce/core/pdp/api/io/BaseXacmlJaxbResultPostprocessor.Factory.html>_.
3. When your implementation class is ready, create a text file `org.ow2.authzforce.core.pdp.api.PdpExtension` in folder `src/main/resources/META-INF/services` (you have to create the folder first) and put the fully qualified name of your implementation class on the first line of this file, like in the example from AuthzForce source code.
4. Run Maven `package` to produce a JAR from the Maven project.

Now you have a Result Postprocessor extension ready for integration into AuthzForce Server, as explained in the next section.

5.3.5.7.5.2 Integrating a Result Postprocessor extension into AuthzForce Server

This section assumes you have a Combining Algorithm extension in form of a JAR, typically produced by the process described in the previous section. You may use AuthzForce PDP Core Tests JAR if you only wish to test the examples in this documentation. This JAR is available on Maven Central: groupId= `org.ow2.authzforce`, artifactId= `authzforce-ce-core-pdp-testutils`, version= `17.1.0`.

The steps to integrate the extension into the AuthzForce Server go as follows:

1. Make the JAR - and any extra dependency - visible from the AuthzForce webapp in Tomcat. One way to do it consists to copy the JAR (e.g. `authzforce-ce-core-pdp-testutils-13.3.1.jar` in our example) into `/opt/authzforce-ce-server/webapp/WEB-INF/lib`. For other ways, please refer to Tomcat HowTo.
2. Finally, restart Tomcat to apply changes.

5.3.5.7.5.3 Enabling a Result Postprocessor extension on a domain

Once you have deployed the extension on Authzforce, following previous instructions, you are ready to enable it on a specific domain's PDP by updating the PDP properties with an enabled `feature` of type `urn:ow2:authzforce:feature-type:pdp:result-postproc`. The following example enables Authzforce combined decision result postprocessor (implementing the feature `urn:oasis:names:tc:xacml:3.0:profile:multiple:combined-decision` from XACML v3.0 Multiple Decision Profile Version 1.0 for testing) on the PDP, provided that the AuthzForce PDP Core Tests JAR has been deployed (see previous section):

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/pdp.properties
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdpPropertiesUpdate xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
 <feature
  type="urn:ow2:authzforce:feature-type:pdp:result-postproc"
  enabled="true">urn:ow2:authzforce:feature:pdp:result-postproc:multiple:test-combined-decision</feature>
 <rootPolicyRefExpression>root</rootPolicyRefExpression>
</pdpPropertiesUpdate>
```

5.3.5.7.6 Attribute Providers

The API allows to manage PDP attribute providers. These are PDP extensions that enable the PDP to get attributes from other sources than PEPs' requests. Such sources may be remote services, databases, etc. The AuthzForce Server distribution does not provide attribute providers out of the box, but allows you to plug in custom-made one(s) from your own invention or from third parties. The AuthzForce project also provides a separate Attribute Provider example, for testing and documentation purposes only. If you wish to make your own attribute provider, read on the next section. If you wish to test the example provided by AuthzForce or have another one ready for use, you may jump to the section *Integrating an Attribute Provider into AuthzForce Server*.

5.3.5.7.6.1 Making an Attribute Provider

The steps to make your own PDP Attribute Provider extension for AuthzForce go as follows:

1. Create a Maven project with `jar` packaging type.
2. Create an XML schema file with `.xsd` extension in the `src/main/resources` folder of your Maven project. Make sure this filename is potentially unique on a Java classpath, like your usual Java class names. One way to make sure is to use a filename prefix following the same conventions as the Java package naming conventions. In this schema file, define an XML type for your attribute provider configuration format. This type must extend `AbstractAttributeProvider` from namespace `http://authzforce.github.io/xmlns/pdp/ext/3`. You may use the schema of AuthzForce TestAttributeProvider (used for AuthzForce unit tests only) as an example. In this example, the XSD filename is `org.ow2.authzforce.core.pdp.testutil.ext.xsd` and the defined XML type extending `AbstractAttributeProvider` is `TestAttributeProviderDescriptor`.
3. Copy the files `bindings.xjb` and `catalog.xml` from AuthzForce source code into the `src/main/jaxb` folder (you have to create this folder first) of your Maven project.
4. Add the following Maven dependency and build plugin configuration to your Maven POM:

```
<dependency>
 <groupId>org.ow2.authzforce</groupId>
 <artifactId>authzforce-ce-core-pdp-api</artifactId>
 <version>18.0.1</version> <scope>provided</scope>
</dependency>
...
<build>
 ...
 <plugins>
  <plugin>
   <groupId>org.jvnet.jaxb2.maven2</groupId>
   <artifactId>maven-jaxb2-plugin</artifactId>
   <version>0.14.0</version>
   <configuration>
    <debug>false</debug>
    <strict>false</strict>
    <verbose>true</verbose>
    <removeOldOutput>true</removeOldOutput>
    <extension>true</extension>
    <useDependenciesAsEpisodes>true</useDependenciesAsEpisodes>
    <catalog>src/main/jaxb/catalog.xml</catalog>
    <bindingDirectory>src/main/jaxb</bindingDirectory>
    <schemaDirectory>src/main/resources</schemaDirectory>
   </configuration>
   <executions>
    <execution>
     <id>jaxb-generate-compile-sources</id>
     <phase>generate-sources</phase>
     <goals>
      <goal>generate</goal>
     </goals>
    </execution>
   </executions>
  </plugin>
  ...
 </plugins>
</build>
...
```

5. Run Maven `generate-sources`. This will generate the JAXB-annotated class(es) from the XML schema into the folder `target/generated-sources/xjc`, one of which corresponds to your attribute provider XML type defined in the second step, therefore has the same name and also extends `org.ow2.authzforce.xmlns.pdp.ext.AbstractAttributeProvider` class corresponding to `AbstractAttributeProvider` type in the XML schema. For example, in the case of the AuthzForce *Test AttributeProvider* aforementioned, the corresponding generated class is `org.ow2.authzforce.core.pdp.testutil.ext.xmlns.TestAttributeProviderDescriptor`. In your case and in general, we will refer to it as your *Attribute Provider Model class*.

6. Create your Attribute Provider factory and concrete implementation class (as in the *Factory* design pattern). The factory class must be public, and extend CloseableNamedAttributeProvider.FactoryBuilder<APM>, where `APM` stands for your *Attribute Provider Model Class*; and the factory class must have a public no-argument constructor or no constructor. You may use the AuthzForce TestAttributeProvider implementation class (used for AuthzForce unit tests only) as an example. In this example, the static nested class `Factory` is the one extending `CloseableNamedAttributeProvider.FactoryBuilder<TestAttributeProviderDescriptor>`. Such a class has a factory method `getInstance(APM config, ...)` (`getInstance(TestAttributeProviderDescriptor conf, ...)` in the example) that, from an instance of your `APM` representing the XML input (`TestAttributeProviderDescriptor` in the example), creates an instance of your Attribute Provider implementation class (`TestAttributeProvider` in the example). Indeed, your Attribute Provider implementation class must implement the interface CloseableNamedAttributeProvider. To facilitate the implementation process, instead of implementing this interface directly, you should extend BaseNamedAttributeProvider in your implementation class, whenever possible. This class already implements the required interface. There

are cases where it is not possible; for instance, since BaseNamedAttributeProvider is an abstract class, if your implementation needs to extend another abstract class, you have no choice but to implement the interface directly, because a Java class cannot extend multiple abstract classes. In any case, as mandated by the interface, your implementation class must implement the method `get(attributeFQN, datatype, context)` in charge of actually retrieving the extra attributes (`TestAttributeProvider#get(...)` in the example). The `attributeFQN` identifies an XACML attribute's fully qualified name, i.e. category, ID and optional Issuer that the PDP is requesting from your attribute provider; the `datatype` is the expected attribute datatype; and `context` is the request context, including the content from the current XACML Request and possibly extra attributes retrieved so far by other Attribute Providers.

7. When your Attribute Provider implementation class is ready, you create a configuration file to enable AuthzForce extension manager to discover this new extension dynamically at runtime. AuthzForce extension mechanism is based on Java native extension mechanism. In this regard, the PdpExtension interface is the SPI, and your AttributeProvider implementation must be registered as one of the *Service Provider* for this SPI. In short, all you have to do is create a configuration file `org.ow2.authzforce.core.pdp.api.PdpExtension` in folder `src/main/resources/META-INF/services` (you may have to create the folder first) and put the fully qualified name of your implementation class on the first line of this file (or a new line if there are others already there), like in the example from AuthzForce source code. More info.
8. Run Maven `package` to produce a JAR from the Maven project.

Now you have an Attribute Provider extension ready for integration into AuthzForce Server, as explained in the next section.

5.3.5.7.6.2 Integrating an Attribute Provider into AuthzForce Server

This section assumes you have an Attribute Provider extension in form of a JAR, typically produced by the process in the previous section. You may use AuthzForce PDP Core Tests JAR if you only wish to test the examples in this documentation. This JAR is available on Maven Central: groupId= `org.ow2.authzforce`, artifactId= `authzforce-ce-core-pdp-testutils`, version= `17.1.0`.

The steps to integrate the extension into the AuthzForce Server go as follows:

1. Make the JAR - and any extra dependency - visible from the AuthzForce webapp in Tomcat. One way to do it consists to copy the JAR (e.g. `authzforce-ce-core-pdp-testutils-13.3.1.jar` in our example) into `/opt/authzforce-ce-server/webapp/WEB-INF/lib`. For other ways, please refer to Tomcat HowTo.
2. Import your attribute provider XML schema in the XML schema file `/opt/authzforce-ce-server/conf/authzforce-ext.xsd`, using `namespace` **only** (no `schemaLocation`), like in the example from Authzforce code with this schema import for AuthzForce `TestAttributeProvider`:

```
<xs:import namespace="http://authzforce.github.io/core/xmlns/test/3" />
```

3. Add a `uri` element to XML catalog file `/opt/authzforce-ce-server/conf/catalog.xml`, with your attribute Provider XML namespace as `name` attribute value, and, the location of your XML schema file within the JAR, as `uri` attribute value, prefixed by `classpath:`. For example, in the sample XML catalog from Authzforce source code, we add the following line for AuthzForce `TestAttributeProvider`:

```
<uri name="http://authzforce.github.io/core/xmlns/test/3" uri="classpath:org.ow2.authzforce.core.pdp.
↪testutil.ext.xsd"/>
```

4. Finally, restart Tomcat to apply changes.

5.3.5.7.6.3 Managing attribute providers configuration

Once you have deployed a new attribute provider extension on Authzforce, following previous instructions, you are ready to use it on a domain:

- Method: PUT
- Path: `/domains/{domainId}/pap/attribute.providers`
- Headers:
  - Content-Type: `application/xml; charset=UTF-8`
  - Accept: `application/xml; charset=UTF-8`
- Body: new attribute providers.

For example, this request instantiates a specific `TestAttributeProvider` configuration on domain `iMnxv7sDEeWFwqVFFMDLTQ` (as mentioned in the previous section, `TestAttributeProvider` is merely an example for testing and documentation purposes, it is not available in a default installation of Authzforce):

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/attribute.providers
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<attributeProviders
 xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
 xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
 <attributeProvider
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:test="http://authzforce.github.io/core/xmlns/test/3"
  xsi:type="test:TestAttributeProviderDescriptor" id="test">
  <xacml:Attributes
   Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
   <xacml:Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:example:attribute:role"
    IncludeInResult="false">
    <xacml:AttributeValue
     DataType="http://www.w3.org/2001/XMLSchema#string">Physician</xacml:AttributeValue>
   </xacml:Attribute>
  </xacml:Attributes>
 </attributeProvider>
</attributeProviders>
```

The response is the new attribute provider configuration from the request.

In this second example, we disable all PDP attribute providers of domain `iMnxv7sDEeWFwqVFFMDLTQ` by sending an empty element:

```
PUT /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/attribute.providers
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<attributeProviders xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5" />
```

Finally, you may get the current attribute providers anytime as follows:

- Method: GET
- Path: `/domains/{domainId}/pap/attribute.providers`
- Headers:

- Accept: `application/xml; charset=UTF-8`

For example, this request gets the PDP attribute providers of domain `iMnxv7sDEeWFwqVFFMDLTQ`:

```
GET /domains/iMnxv7sDEeWFwqVFFMDLTQ/pap/attribute.providers
HTTP/1.1
Accept: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<attributeProviders xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
  ...
</attributeProviders>
```

5.3.6 Policy Decision API

The PDP API returns an authorization decision based on the currently enforced policy, access control attributes provided in the request and possibly other attributes resolved by the PDP itself. The Authorization decision is typically `Permit` or `Deny`. The PDP is able to resolve extra attributes not provided directly in the request, such as the current date/time (environment attribute).

The PDP provides an HTTP RESTful API for requesting authorization decisions, that complies with test assertions `urn:oasis:names:tc:xacml:3.0:profile:rest:assertion:pdp:xacml:status`, `urn:oasis:names:tc:xacml:3.0:profile:rest:assertion:pdp:xacml:body` and `urn:oasis:names:tc:xacml:3.0:profile:rest:assertion:pdp:xacml:invalid` of REST Profile of XACML v3.0 Version 1.0.

The HTTP request must be formatted as follows:

- Method: POST
- Path: `/domains/{domainId}/pdp`
- Headers:
  - Content-Type: `application/xml; charset=UTF-8`
  - Accept: `application/xml; charset=UTF-8`
- Body: XACML Request as defined in the XACML 3.0 schema.

The HTTP response body is a XACML Response as defined in the XACML 3.0 schema.

Example of request given below:

```
POST /domains/iMnxv7sDEeWFwqVFFMDLTQ/pdp
HTTP/1.1
Accept: application/xml; charset=UTF-8
Content-Type: application/xml; charset=UTF-8

<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 CombinedDecision="false" ReturnPolicyIdList="false">
 <Attributes
  Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
  <Attribute
   AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
   IncludeInResult="false">
   <AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">joe</AttributeValue>
  </Attribute>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
   IncludeInResult="false"> <AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">Manager</AttributeValue>
```

(continues on next page)

(continued from previous page)

```
  </Attribute>
 </Attributes>
 <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
  <Attribute
   AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
   IncludeInResult="false">
   <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">MissionManagementApp</AttributeValue>
  </Attribute>
  <Attribute
   AttributeId="urn:thales:xacml:2.0:resource:sub-resource-id" IncludeInResult="false">
   <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">Team</AttributeValue>
  </Attribute>
 </Attributes>
 <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
  <Attribute
   AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
   IncludeInResult="false">
   <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">manage</AttributeValue>
  </Attribute>
 </Attributes>
 <Attributes
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment" />
</Request>
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
 <Result>
     <Decision>Permit</Decision>
 </Result>
</Response>
```

If the XACML request was invalid (invalid format), an error 400 is returned.

5.3.7 Fast Infoset

Fast Infoset is an ITU-T/ISO standard for representing XML (XML Information Set to be accurate) using binary encodings, designed for use cases to provide smaller encoding sizes and faster processing than a W3C XML representation as text. The open source Fast Infoset project provide some performance results and more information about the standardisation status. There are several use cases at the origin of Fast Infoset. A major one comes from the Web3D consortium that is responsible for open standards in real-time 3D communication, and that adopted Fast Infoset for the serialization and compression of X3D documents. X3D is a standard for representing 3D scenes and objects using XML.

AuthzForce Server API offers experimental support for Fast Infoset (use with caution). This feature is disabled by default and needs to be enabled explicitly by the administrator as told in the *Fast Infoset mode*. When it is enabled, provided that your API client supports Fast Infoset as well, you may use Fast Infoset on the server API by replacing the media type `application/xml` with `application/fastinfoset` in your API request headers (*Accept*/*Content-Type*).

5.3.8 Integration with the IdM and PEP Proxy GEs (e.g. for OAuth)

AuthzForce integrates with the Identity Management (KeyRock) and PEP Proxy GE (Wilma) reference implementations. For an overview of the main interactions, please refer to the Basic and Advanced sections of Wilma programmer guide.

After you installed and configured KeyRock, to connect it to Authzforce, you modify the properties with names prefixed by `ACCESS_CONTROL_` in the configuration file `fiware-idm/horizon/openstack_dashboard/local/local_settings.py` (example on KeyRock Github repository) according to your AuthzForce instance properties. For example:

```
// ACCESS CONTROL GE
// URL to Authzforce server (http(s)://HOST:PORT)
ACCESS_CONTROL_URL = 'http://127.0.0.1:8080'
// Magic key, required only if securing the AZF with a PEP Proxy
ACCESS_CONTROL_MAGIC_KEY = 'undefined'
```

WARNING: If you are using KeyRock v5.3.0 or older, you also have to change the content of IDM's template file `openstack_dashboard/templates/access_control/policy_properties.xacml` to this (basically the only change consists to remove the `ns2` namespace prefix):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdpPropertiesUpdate xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5">
 <rootPolicyRefExpression>{{ policy_id }}</rootPolicyRefExpression>
</pdpPropertiesUpdate>
```

Then restart the IdM to apply changes, and go to IdM web interface, and check that the permissions and roles are well configured for your application. You may have to 'trigger' the policy generation in IdM by going to your application > *Manage roles* and click *Save* to trigger the XACML generation. More information in KeyRock installation and administration guide.

Then, after you installed and configured Wilma, to connect it to Authzforce, you modify the settings in `config.azf` object of configuration file `config.js` (example) according to your AuthzForce instance properties. More information in Wilma installation and administration guide.

5.3.9 Software Libraries for clients of AuthzForce or other Authorization PDP GEis

The full API (RESTful) is described by a document written in the Web Application Description Language format (WADL) and associated XML schema files available in Authzforce rest-api-model project files. Therefore, you can use any WADL-supporting REST framework for clients; for instance in Java: Jersey, Apache CXF. From that, you can use WADL-to-code generators to generate your client code. For example in Java, 'wadl2java' tools allow to generate code for JAX-RS compatible frameworks such as Apache CXF and Jersey. Actually, we can provide a CXF-based Java library created with this tool to facilitate the development of clients.