

PEP Engine Design Document

EOEPCA.SDD.xxx

TVUK System Team

Version 0.1, dd/mm/yyyy:

PEP Engine Design Document

| | |
|---|----|
| 1. Introduction | 2 |
| 1.1. Purpose and Scope | 2 |
| 1.2. Structure of the Document | 2 |
| 1.3. Reference Documents | 2 |
| 1.4. Terminology | 4 |
| 1.5. Glossary | 9 |
| 2. Overview | 11 |
| 2.1. Building Block Overview | 11 |
| 2.1.1. Initialization flow | 12 |
| 2.2. External Interfaces | 13 |
| 2.2.1. Exposed Interfaces | 13 |
| 2.2.1.1. HTTP(S) (Reverse Proxy Listener) | 13 |
| 2.2.1.2. HTTP(S) (to Resource Server) | 13 |
| 2.2.2. Consumed Interfaces | 13 |
| 2.2.2.1. UMA (to Login Service) | 13 |
| 2.2.2.2. OIDC (to Login Service) | 13 |
| 2.2.2.3. SCIM (to Login Service) | 14 |
| 2.2.2.4. Resource API | 14 |
| 2.2.2.5. Policy API (to Policy Decision Point) | 14 |
| 2.3. Required resources | 14 |
| 2.3.1. Software | 15 |
| 2.4. Static Architecture | 15 |
| 2.5. Use cases | 16 |
| 2.5.1. PEP-UC-001: Self Authentication & Registration | 17 |
| 2.5.2. PEP-UC-002: Ticket generation | 17 |
| 2.5.3. PEP-UC-003: Resource protection & RPT validation | 18 |
| 2.5.4. PEP-UC-004: Request Forwarding with JWT header | 18 |
| 2.5.5. PEP-UC-005: Resource Management | 19 |
| 2.5.6. PEP-UC-006: Default protection of resources | 19 |
| 2.5.7. PEP-UC-007: Policy Enforcement Point API | 19 |
| 3. Design | 20 |
| 3.1. Building Block Design | 20 |
| 3.2. Reverse Proxy Service | 20 |
| 3.2.1. Overview and Purpose | 20 |
| 3.2.2. Software Reuse and Dependencies | 20 |
| 3.2.3. Interfaces | 21 |
| 3.2.4. Data | 21 |
| 3.2.4.1. Configuration | 21 |

| | |
|---|----|
| 3.2.4.1.1. Default HTTP scopes | 22 |
| 3.2.4.1.2. Default resources registration | 22 |
| 3.2.4.2. Data flow | 23 |
| 3.2.5. Extensibility | 23 |
| 3.2.6. Applicable Resources | 24 |
| 3.3. Resource Repository | 24 |
| 3.3.1. Overview and Purpose | 24 |
| 3.3.2. Data flow | 25 |
| 3.3.3. Applicable Resources | 27 |
| 3.4. Resource default Protection Policy | 27 |
| 3.4.1. Overview and Purpose | 27 |
| 3.4.2. Data flow | 27 |
| 3.4.3. Applicable Resources | 27 |
| 3.5. Logging | 27 |
| 3.5.1. Design | 27 |
| 3.5.2. Log message format | 28 |
| 3.5.3. Log message codes | 28 |
| 4. User Story Traceability | 30 |

| | |
|---|---|
| COMMENTS and ISSUES If you would like to raise comments or issues on this document, please do so by raising an Issue at the following URL https://github.com/EOEPCA/um-pep-engine/issues . | PDF This document is available in PDF format here . |
| EUROPEAN SPACE AGENCY CONTRACT REPORT The work described in this report was done under ESA contract. Responsibility for the contents resides in the author or organisation that prepared it. | TELESPAZIO VEGA UK Ltd 350 Capability Green, Luton, Bedfordshire, LU1 3LU, United Kingdom. Tel: +44 (0)1582 399000 www.telespazio-vega.com |

AMENDMENT HISTORY

This document shall be amended by releasing a new edition of the document in its entirety. The Amendment Record Sheet below records the history and issue status of this document.

Table 1. Amendment Record Sheet

| ISSUE | DATE | REASON |
|-------|------------|---------------------------|
| 0.1 | dd/mm/yyyy | Initial in-progress draft |

Chapter 1. Introduction

1.1. Purpose and Scope

This document presents the PEP Engine Design for the Common Architecture.

1.2. Structure of the Document

Section 2 - **Overview**

Provides an overview of the PEP Engine component, within the context of the wider Common Architecture design.

Section 3 - **Design**

Provides the design of the PEP Engine component.

1.3. Reference Documents

The following is a list of Reference Documents with a direct bearing on the content of this document.

| Reference | Document Details | Version |
|-----------------|---|--------------------------|
| [EOEPCA-UC] | EOEPCA - Use Case Analysis EOEPCA.TN.005 https://eoezca.github.io/use-case-analysis | Issue 1.0, 02/08/2019 |
| [EP-FM] | Exploitation Platform - Functional Model, ESA-EOPSDP-TN-17-050 | Issue 1.0, 30/11/2017 |
| [TEP-OA] | Thematic Exploitation Platform Open Architecture, EMSS-EOPS-TN-17-002 | Issue 1, 12/12/2017 |
| [WPS-T] | OGC Testbed-14: WPS-T Engineering Report, OGC 18-036r1, http://docs.opengeospatial.org/per/18-036r1.html | 18-036r1, 07/02/2019 |
| [WPS-REST-JSON] | OGC WPS 2.0 REST/JSON Binding Extension, Draft, OGC 18-062, https://raw.githubusercontent.com/opengeospatial/wps-rest-binding/develop/docs/18-062.pdf | 1.0-draft |
| [CWL] | Common Workflow Language Specifications, https://www.commonwl.org/v1.0/ | v1.0.2 |

| Reference | Document Details | Version |
|-------------------|--|-------------------------|
| [TB13-AP] | OGC Testbed-13, EP Application Package Engineering Report, OGC 17-023, http://docs.opengeospatial.org/per/17-023.html | 17-023, 30/01/2018 |
| [TB13-ADES] | OGC Testbed-13, Application Deployment and Execution Service Engineering Report, OGC 17-024, http://docs.opengeospatial.org/per/17-024.html | 17-024, 11/01/2018 |
| [TB14-AP] | OGC Testbed-14, Application Package Engineering Report, OGC 18-049r1, http://docs.opengeospatial.org/per/18-049r1.html | 18-049r1, 07/02/2019 |
| [TB14-ADES] | OGC Testbed-14, ADES & EMS Results and Best Practices Engineering Report, OGC 18-050r1, http://docs.opengeospatial.org/per/18-050r1.html | 18-050r1, 08/02/2019 |
| [OS-GEO-TIME] | OpenSearch GEO: OpenSearch Geo and Time Extensions, OGC 10-032r8, http://www.opengeospatial.org/standards/opensearchgeo | 10-032r8, 14/04/2014 |
| [OS-EO] | OpenSearch EO: OGC OpenSearch Extension for Earth Observation, OGC 13-026r9, http://docs.opengeospatial.org/is/13-026r8/13-026r8.html | 13-026r9, 16/12/2016 |
| [GEOJSON-LD] | OGC EO Dataset Metadata GeoJSON(-LD) Encoding Standard, OGC 17-003r1/17-084 | 17-003r1/17-084 |
| [GEOJSON-LD-RESP] | OGC OpenSearch-EO GeoJSON(-LD) Response Encoding Standard, OGC 17-047 | 17-047 |
| [PCI-DSS] | The Payment Card Industry Data Security Standard, https://www.pcisecuritystandards.org/document_library?category=pcidss&document=pci_dss | v3.2.1 |
| [CEOS-OS-BP] | CEOS OpenSearch Best Practise, http://ceos.org/ourwork/workinggroups/wgiss/access/opensearch/ | v1.2, 13/06/2017 |
| [OIDC] | OpenID Connect Core 1.0, https://openid.net/specs/openid-connect-core-1_0.html | v1.0, 08/11/2014 |

| Reference | Document Details | Version |
|------------|--|---------------------------|
| [OGC-CSW] | OGC Catalogue Services 3.0 Specification - HTTP Protocol Binding (Catalogue Services for the Web), OGC 12-176r7, http://docs.openeospatial.org/is/12-176r7/12-176r7.html | v3.0, 10/06/2016 |
| [OGC-WMS] | OGC Web Map Server Implementation Specification, OGC 06-042, http://portal.openeospatial.org/files/?artifact_id=14416 | v1.3.0, 05/03/2006 |
| [OGC-WMTS] | OGC Web Map Tile Service Implementation Standard, OGC 07-057r7, http://portal.openeospatial.org/files/?artifact_id=35326 | v1.0.0, 06/04/2010 |
| [OGC-WFS] | OGC Web Feature Service 2.0 Interface Standard – With Corrigendum, OGC 09-025r2, http://docs.openeospatial.org/is/09-025r2/09-025r2.html | v2.0.2, 10/07/2014 |
| [OGC-WCS] | OGC Web Coverage Service (WCS) 2.1 Interface Standard - Core, OGC 17-089r1, http://docs.openeospatial.org/is/17-089r1/17-089r1.html | v2.1, 16/08/2018 |
| [OGC-WCPS] | Web Coverage Processing Service (WCPS) Language Interface Standard, OGC 08-068r2, http://portal.openeospatial.org/files/?artifact_id=32319 | v1.0.0, 25/03/2009 |
| [AWS-S3] | Amazon Simple Storage Service REST API, https://docs.aws.amazon.com/AmazonS3/latest/API | API Version 2006-03-01 |

1.4. Terminology

The following terms are used in the Master System Design.

| Term | Meaning |
|-----------------|--|
| Admin | User with administrative capability on the EP |
| Algorithm | A self-contained set of operations to be performed, typically to achieve a desired data manipulation. The algorithm must be implemented (codified) for deployment and execution on the platform. |
| Analysis Result | The <i>Products</i> produced as output of an <i>Interactive Application</i> analysis session. |

| Term | Meaning |
|---|--|
| Analytics | A set of activities aimed to discover, interpret and communicate meaningful patterns within the data. Analytics considered here are performed manually (or in a semi-automatic way) on-line with the aid of <i>Interactive Applications</i> . |
| Application Artefact | The 'software' component that provides the execution unit of the <i>Application Package</i> . |
| Application Deployment and Execution Service (ADES) | WPS-T (REST/JSON) service that incorporates the Docker execution engine, and is responsible for the execution of the processing service (as a WPS request) within the 'target' Exploitation Platform. |
| Application Descriptor | A file that provides the metadata part of the <i>Application Package</i> . Provides all the metadata required to accommodate the processor within the WPS service and make it available for execution. |
| Application Package | A platform independent and self-contained representation of a software item, providing executable, metadata and dependencies such that it can be deployed to and executed within an Exploitation Platform. Comprises the <i>Application Descriptor</i> and the <i>Application Artefact</i> . |
| Bulk Processing | Execution of a <i>Processing Service</i> on large amounts of data specified by AOI and TOI. |
| Code | The codification of an algorithm performed with a given programming language - compiled to Software or directly executed (interpreted) within the platform. |
| Compute Platform | The Platform on which execution occurs (this may differ from the Host or Home platform where federated processing is happening) |
| Consumer | User accessing existing services/products within the EP. Consumers may be scientific/research or commercial, and may or may not be experts of the domain |
| Data Access Library | An abstraction of the interface to the data layer of the resource tier. The library provides bindings for common languages (including python, Javascript) and presents a common object model to the code. |
| Development | The act of building new products/services/applications to be exposed within the platform and made available for users to conduct exploitation activities. Development may be performed inside or outside of the platform. If performed outside, an integration activity will be required to accommodate the developed service so that it is exposed within the platform. |
| Discovery | User finds products/services of interest to them based upon search criteria. |
| Execution | The act to start a <i>Processing Service</i> or an <i>Interactive Application</i> . |

| Term | Meaning |
|------------------------------------|--|
| Execution Management Service (EMS) | The EMS is responsible for the orchestration of workflows, including the possibility of steps running on other (remote) platforms, and the on-demand deployment of processors to local/remote ADES as required. |
| Expert | User developing and integrating added-value to the EP (Scientific Researcher or Service Developer) |
| Exploitation Tier | The Exploitation Tier represents the end-users who exploit the services of the platform to perform analysis, or using high-level applications built-in on top of the platform's services |
| External Application | An application or script that is developed and executed outside of the Exploitation Platform, but is able to use the data/services of the EP via a programmatic interface (API). |
| Guest | An unregistered User or an unauthenticated Consumer with limited access to the EP's services |
| Home Platform | The Platform on which a User is based or from which an action was initiated by a User |
| Host Platform | The Platform through which a Resource has been published |
| Identity Provider (IdP) | The source for validating user identity in a federated identity system, (user authentication as a service). |
| Interactive Application | A stand-alone application provided within the exploitation platform for on-line hosted processing. Provides an interactive interface through which the user is able to conduct their analysis of the data, producing <i>Analysis Results</i> as output. Interactive Applications include at least the following types: console application, web application (rich browser interface), remote desktop to a hosted VM. |
| Interactive Console Application | A simple <i>Interactive Application</i> for analysis in which a console interface to a platform-hosted terminal is provided to the user. The console interface can be provided through the user's browser session or through a remote SSH connection. |
| Interactive Remote Desktop | An Interactive Application for analysis provided as a remote desktop session to an OS-session (or directly to a 'native' application) on the exploitation platform. The user will have access to a number of applications within the hosted OS. The remote desktop session is provided through the user's web browser. |
| Interactive Web Application | An Interactive Application for analysis provided as a rich user interface through the user's web browser. |
| Key-Value Pair | A key-value pair (KVP) is an abstract data type that includes a group of key identifiers and a set of associated values. Key-value pairs are frequently used in lookup tables, hash tables and configuration files. |
| Kubernetes (K8s) | Container orchestration system for automating application deployment, scaling and management. |

| Term | Meaning |
|------------------------------|---|
| Login Service | An encapsulation of Authenticated Login provision within the Exploitation Platform context. The Login Service is an OpenID Connect Provider that is used purely for authentication. It acts as a Relying Party in flows with external IdPs to obtain access to the user's identity. |
| EO Network of Resources | The coordinated collection of European EO resources (platforms, data sources, etc.). |
| Object Store | A computer data storage architecture that manages data as objects. Each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier. |
| On-demand Processing Service | A <i>Processing Service</i> whose execution is initiated directly by the user on an ad-hoc basis. |
| Platform (EP) | An on-line collection of products, services and tools for exploitation of EO data |
| Platform Tier | The Platform Tier represents the Exploitation Platform and the services it offers to end-users |
| Processing | A set of pre-defined activities that interact to achieve a result. For the exploitation platform, comprises on-line processing to derive data products from input data, conducted by a hosted processing service execution. |
| Processing Result | The <i>Products</i> produced as output of a <i>Processing Service</i> execution. |
| Processing Service | A non-interactive data processing that has a well-defined set of input data types, input parameterisation, producing <i>Processing Results</i> with a well-defined output data type. |
| Products | EO data (commercial and non-commercial) and Value-added products and made available through the EP. <i>It is assumed that the Hosting Environment for the EP makes available an existing supply of EO Data</i> |
| Resource | A entity, such as a Product, Processing Service or Interactive Application, which is of interest to a user, is indexed in a catalogue and can be returned as a single meaningful search result |
| Resource Tier | The Resource Tier represents the hosting infrastructure and provides the EO data, storage and compute upon which the exploitation platform is deployed |
| Reusable Research Object | An encapsulation of some research/analysis that describes all aspects required to reproduce the analysis, including data used, processing performed etc. |
| Scientific Researcher | Expert user with the objective to perform scientific research. Having minimal IT knowledge with no desire to acquire it, they want the effort for the translation of their algorithm into a service/product to be minimised by the platform. |

| Term | Meaning |
|--|--|
| Service Developer | Expert user with the objective to provide a performing, stable and reliable service/product. Having deeper IT knowledge or a willingness to acquire it, they require deeper access to the platform IT functionalities for optimisation of their algorithm. |
| Software | The compilation of code into a binary program to be executed within the platform on-line computing environment. |
| Systematic Processing Service | A <i>Processing Service</i> whose execution is initiated automatically (on behalf of a user), either according to a schedule (routine) or triggered by an event (e.g. arrival of new data). |
| Terms & Conditions (T&Cs) | The obligations that the user agrees to abide by in regard of usage of products/services of the platform. T&Cs are set by the provider of each product/service. |
| Transactional Web Processing Service (WPS-T) | Transactional extension to WPS that allows adhoc deployment / undeployment of user-provided processors. |
| User | An individual using the EP, of any type (Admin/Consumer/Expert/Guest) |
| Value-added products | Products generated from processing services of the EP (or external processing) and made available through the EP. This includes products uploaded to the EP by users and published for collaborative consumption |
| Visualisation | To obtain a visual representation of any data/products held within the platform - presented to the user within their web browser session. |
| Web Coverage Service (WCS) | OGC standard that provides an open specification for sharing raster datasets on the web. |
| Web Coverage Processing Service (WCPS) | OGC standard that defines a protocol-independent language for the extraction, processing, and analysis of multi-dimensional coverages representing sensor, image, or statistics data. |
| Web Feature Service (WFS) | OGC standard that makes geographic feature data (vector geospatial datasets) available on the web. |
| Web Map Service (WMS) | OGC standard that provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases. |
| Web Map Tile Service (WMTS) | OGC standard that provides a simple HTTP interface for requesting map tiles of spatially referenced data using the images with predefined content, extent, and resolution. |
| Web Processing Services (WPS) | OGC standard that defines how a client can request the execution of a process, and how the output from the process is handled. |
| Workspace | A user-scoped 'container' in the EP, in which each user maintains their own links to resources (products and services) that have been collected by a user during their usage of the EP. The workspace acts as the hub for a user's exploitation activities within the EP |

1.5. Glossary

The following acronyms and abbreviations have been used in this report.

| Term | Definition |
|----------|---|
| AAI | Authentication & Authorization Infrastructure |
| ABAC | Attribute Based Access Control |
| ADES | Application Deployment and Execution Service |
| ALFA | Abbreviated Language For Authorization |
| AOI | Area of Interest |
| API | Application Programming Interface |
| CMS | Content Management System |
| CWL | Common Workflow Language |
| DAL | Data Access Library |
| EMS | Execution Management Service |
| EO | Earth Observation |
| EP | Exploitation Platform |
| FUSE | Filesystem in Userspace |
| GeoXACML | Geo-specific extension to the XACML Policy Language |
| IAM | Identity and Access Management |
| IdP | Identity Provider |
| JSON | JavaScript Object Notation |
| K8s | Kubernetes |
| KVP | Key-value Pair |
| M2M | Machine-to-machine |
| OGC | Open Geospatial Consortium |
| PDE | Processor Development Environment |
| PDP | Policy Decision Point |
| PEP | Policy Enforcement Point |
| PIP | Policy Information Point |
| RBAC | Role Based Access Control |
| REST | Representational State Transfer |
| SSH | Secure Shell |
| TOI | Time of Interest |
| UMA | User-Managed Access |

| Term | Definition |
|-------------|---|
| VNC | Virtual Network Computing |
| WCS | Web Coverage Service |
| WCPS | Web Coverage Processing Service |
| WFS | Web Feature Service |
| WMS | Web Map Service |
| WMTS | Web Map Tile Service |
| WPS | Web Processing Service |
| WPS-T | Transactional Web Processing Service |
| XACML | eXtensible Access Control Markup Language |

Chapter 2. Overview

2.1. Building Block Overview



Content Description

This section contains:

- High-Level Description of the Building Block
- Context within EOEPKA

The main functionality of the PEP is to be able to stand between a client and the client's desired resource. By creating this setup, where only the PEP is visible to an external request, we effectively secure whatever is behind the PEP. The PEP will enforce any policy for a resource configured in the Authorization Server, following the UMA 2.0 standard.

When launched, the PEP will answer to all requests that start with the configured path. These answers will come in the form of UMA tickets (if there are no RPT provided, or an invalid one is used).

In case the request is accompanied by a header using a valid RPT in the format **Authorization: Bearer <valid_RPT>**, the PEP will make a request to the resource server, for the resource located exactly at the path requested (minus the configured at config), and return the resource's server answer.

Examples follow, assuming the following conditions:

- path configured for the reverse proxy is `"/pep"`
- the PEP is at `pep.domain.com/pep`
- Resource server is at `remote.server.com`

| Token | Request to PEP | PEP Action | PEP answer |
|---------------------------------------|--|---|--|
| No RPT | pep.domain.com | None (request does not get to PEP endpoint) | None (the PEP doesn't see this request) |
| No RPT | pep.domain.com/pep/thing | Generate ticket for <code>"/thing"</code> | 401 + ticket |
| Valid RPT for <code>"/thing"</code> | pep.domain.com/pep/thing | Request to <code>remote.server.com/thing</code> | Contents of <code>remote.server.com/thing</code> |
| Valid RPT for <code>"/thing"</code> | pep.domain.com/pep/different | Generate ticket for <code>"/different"</code> | 401 + ticket |
| INVALID RPT for <code>"/thing"</code> | pep.domain.com/pep/thing | Generate ticket for <code>"/thing"</code> | 401 + ticket |
| No RPT | pep.domain.com/pep/thing/with/large/path | Generate ticket for <code>"/thing/with/large/path"</code> | 401 + ticket |

| Token | Request to PEP | PEP Action | PEP answer |
|---|--|---|--|
| Valid RPT for "/thing/with/large/path" | pep.domain.com/pep/thing/with/large/path | Request to remote.server.com/thing/with/large/path | Contents of remote.server.com/thing/with/large/path |

To further clarify the flow the PEP uses, you can also take a look at the Data Flow in section [Data flow](#)

2.1.1. Initialization flow



2.2. External Interfaces

2.2.1. Exposed Interfaces

2.2.1.1. HTTP(S) (Reverse Proxy Listener)

An HTTP listener, which can be configured through the config file.

This is the only input interface to interact directly with the PEP from outside, and is managed by the reverse proxy.

The default listener for this interface is `/pep/<path-to-resource>`.

This interface will parse the path and the headers in order to assert authentication and authorization of the client requesting the resource.

2.2.1.2. HTTP(S) (to Resource Server)

The PEP will contact via HTTP with the configured Resource Server whenever a valid request with a valid RPT is done, or whenever RPT is not needed to access the resource.

The PEP will make a request to the RS, and will return the answer verbatim to the client that requested it, effectively acting like a transparent proxy from the client's point of view. This allows the mentioned desired behaviour of being able to protect anything just placing the PEP "in front of" the resource to protect.

2.2.2. Consumed Interfaces

2.2.2.1. UMA (to Login Service)

The PEP will make the requests needed to handle the resources, along with the necessary requests to create tickets and check the validity of RPTs.

The endpoints used for UMA are:

- Resource registration: `/oauth/restv1/host/rsrc/resource_set/<resource-id>`
- Permission: `/oauth/restv1/host/rsrc_pr`
- Introspection: `/oauth/restv1/rpt/status`

2.2.2.2. OIDC (to Login Service)

The PEP uses the OIDC protocol in order to authenticate itself as a valid UMA client, and uses this OIDC client in all UMA-related queries.

These queries are done against the Login Service, and the endpoints used are:

- Token: `/oauth/restv1/token`

2.2.2.3. SCIM (to Login Service)

The PEP has the capability to auto-register itself as a client if there is no client pre-configured from previous starts or previous configuration.

In order to do this, it utilizes the SCIM protocol, and queries the Login Service.

The endpoints used for SCIM are:

- User: /restv1/scim/v2/Users
- Token: /oauth/restv1/token
- Register: /oauth/restv1/register

2.2.2.4. Resource API

The Resource API Endpoints offered by the PEP component are protected based on the unique identifier of the Resource Owner that is adding/removing/editing resources.

The Resource API is protected with OAuth/OIDC in the PEP, expecting any of these tokens:

- JWT id_tokens: in this case the PEP extracts the necessary claims from the JWT uniquely identifying the user (“sub” parameter).
The signature of this token will be verified if the signature verification is enabled in the environment variables.
In case it is enabled, it will be distinguished if the JWT obtained in the header is signed with the internal keys of the platform or the building block.
If the platform signature has been used, it will be verified with the platform endpoint. If the signature is from the PEP block, it will be verified with the public key from the PEP.
In case it is disabled, the signature will not be verified but the other steps above will be performed. The PEP extracts the necessary claims from "sub" parameter.
- OAuth Access Token: in this case the PEP performs a query against the User-Info endpoint, uniquely identifying the user.

The UUID of the End-User will be included as attribute of the Resource description document (extending the data model) upon resource creation (with an “ownership_id” field).

Subsequent requests to the specific Resource ID will perform a JWT or OAuth2.0 check, cross-checking against the “ownership_id” before performing actions and answering back with a 401 Unauthorized if there is no match.

2.2.2.5. Policy API (to Policy Decision Point)

When registering a new resource, the PEP will consume the Policy API to register a default policy with the PDP. The endpoint for this is:

- <pdp_url>/policy

2.3. Required resources



Content Description

This section contains:

- List of HW and SW required resources for the correct functioning of the building Block
- References to open repositories (when applicable)

2.3.1. Software

The following Open-Source Software is required to support the deployment and integration of the Policy Enforcement Point:

- EOEPKA's SCIM Client - <https://github.com/EOEPKA/um-common-scim-client>
- EOEPKA's UMA Client - <https://github.com/EOEPKA/um-common-uma-client>
- EOEPKA's Well Known Handler - <https://github.com/EOEPKA/well-known-handler>
- EOEPKA's Policy Decision Point - <https://github.com/EOEPKA/um-pdp-engine>
- Flask - <https://github.com/pallets/flask>
- MongoDB for Python - <https://pymongo.readthedocs.io/en/stable/index.html>

2.4. Static Architecture



Content Description

This section contains:

- Diagram and description of the major logical components within the Building Block



The next section [Design](#) contains detailed descriptions and references needed to understand the intricacies of this component.

2.5. Use cases



Content Description

This section contains:

- Diagrams and definition of the use cases covered by this Building Block



2.5.1. PEP-UC-001: Self Authentication & Registration

(Represented in the above graph by the request to Login Service in the section "PEP Authentication")

The PEP has an internal UMA Client used for all the necessary UMA requests. This client is completely auto-managed even to the point of self-registration, so no pre-configuration is needed in order to run a PEP instance.

2.5.2. PEP-UC-002: Ticket generation

(Represented in the above graph by the request to Login Service called "Request for ticket")

The PEP generates appropriate tickets for access attempts to a resource, which can later be consumed and checked by the Authorization Server in order to give proper clearance to access that same resource.

Ticket generation as per the UMA 2.0 standard, are only valid for that requested it and for that specific resource. When the ticket is generated, this ticket will contains a specific scope which it depends on the

HTTP verb obtained. This scope in the ticket could be 'protected_read' or 'protected_write'.

2.5.3. PEP-UC-003: Resource protection & RPT validation

The PEP when presented with an RPT in an **Authorization** HTTP header, will check the validity of this token for the requested resource. This token is valid for a limited time, for a specific user, and for a specific resource. This makes attacks via copying an RPT extremely inefficient for an attacker. The validation of the rpt token was extended by including a new parameter that allows to establish the number of uses of the rpt. To store the rpt that will be used will be stored in the database with the number of uses that the rpt has.

The PEP will only protect the resources that it recognizes as such. This means that, even without an RPT, the PEP will allow a client to pass-through directly to the resource server if there is no identified resource that matches what the client is requesting.

This behaviour, which is analogous to a blacklist approach (we only deny access a priori of a bunch of resources), can be easily switched to a "whitelist" with simple changes in the code.

On the other hand, this baseline functionality is desirable to allow PEP-chaining, and allows for more complex workflows in the future.

2.5.4. PEP-UC-004: Request Forwarding with JWT header

After validating the RPT we proceed to verify the signature of the JWT if the signature verification is enabled in the environment variables and then we pass the JWT to the request header in the request to the resource server.

If the header has a RPT as token we make a call to the introspection endpoint (/oauth/restv1/rpt/status) passing through parameters the RPT and the pat. Returning a JSON with the information for that token, called claims, where the user name can appear, for example.

Then we proceed to generate this JSON to the format of JWT using an asymmetric cryptography, in this case using RSA with a private key. And then pass this JWT as a header in the request to the resource server.

If the verification of the signature for the JWT is disabled, the code will do the introspection steps in the case of a RPT, and then will add the JWT to the request header without verifying the token signature.



This diagram covers the following use cases:

2.5.5. PEP-UC-005: Resource Management

The PEP allows for calls done on specific API endpoints for resource management. The currently available options are:

- * Create resource
- * Get resource (based on resource ID)
- * Get all available resources (based on user id)
- * Modify resource
- * Delete resource

2.5.6. PEP-UC-006: Default protection of resources

A new dynamically (via CRUD operations) registered resource will have a default protection policy indexed to it. For this goal, the PEP will contact the Policy Decision Point in order to register said policy, associate it with this resource and establish user ownership.

2.5.7. PEP-UC-007: Policy Enforcement Point API

The current implemented functionalities can be consulted through a specific OpenAPI webpage, available at the PEP level.

Chapter 3. Design

3.1. Building Block Design

Content Description

This section contains:



- A concise breakdown of the Building Block in several independent services (when applicable). For each component, the following subsections are added:
 - Overview and purpose: indicating the functionality covered by the component
 - SW Reuse and Dependencies: indicating reuse of third party open source solutions (if any) and any pre-required Dependencies
 - Interfaces: both internal to the building block and those exposed externally
 - Data: Data usage of the building block, data flow and any GDPR concerns should be addressed here
 - Applicable Resources: links and references to (Reference Docs), and repositories.

When a breakdown is necessary, a general overview of the building block can be given. On the contrary, no breakdown indicates a single component development with the same expected sections.

3.2. Reverse Proxy Service

3.2.1. Overview and Purpose

The Flask-based reverse proxy serves as the interface for input queries. This reverse proxy is in charge of receiving the queries and returning the appropriate HTTP response.

3.2.2. Software Reuse and Dependencies

All requirements for the executing of the reverse proxy are found under `src/requirements.txt`, and expect Python 3.6.9 or greater to work.

The most important are:

- **EOEPCA-SCIM**: Used to auto-register itself as a client to the Auth. Server upon startup
- **EOEPCA-UMA**: Used as the backbone of the PEP, allows for generation of tickets, verification of RPTs, and any other UMA-related action.
- **EOEPCA-OIDC**: Used to generate PAT tokens, validate OAuth tokens and JWTs.
- **WellKnownHandler**: Used to dynamically check the configuration of the Authorization Server on each execution. For example, it can get the needed endpoints for any API the PEP needs, such

as the token request for OIDC.

- **Flask:** External to EOEPKA's project, this library allows the PEP to create the actual reverse proxy, sending and receiving custom requests.
- **MongoDB:** Used to store the resources and user identifications

3.2.3. Interfaces

This component doesn't have any internal interfaces. For a reference of external interfaces see [External Interfaces](#) on Section 2 [Overview](#)

3.2.4. Data

3.2.4.1. Configuration

The PEP gets all its configuration from the files located under `config/*.json`.

The parameters that are accepted, and their meaning, are as follows. For the `config.json` file:

- **realm:** 'realm' parameter answered for each UMA ticket. Default is "eoezca"
- **auth_server_url:** complete url (with "https") of the Authorization server.
- **service_host:** Host for the proxy to listen on. For example, "0.0.0.0" will listen on all interfaces
- **proxy_service_port:** Port for the proxy to listen on. By default, 5566. Keep in mind you will have to edit the docker file and/or kubernetes yaml file in order for all the port forwarding to work.
- **resources_service_port:** Port for the resources to listen on. By default, 5576. Keep in mind you will have to edit the docker file and/or kubernetes yaml file in order for all the port forwarding to work.
- **s_margin_rpt_valid:** An integer representing how many seconds of "margin" do we want when checking RPT. For example, using 5 will make sure the provided RPT is valid now AND AT LEAST in the next 5 seconds.
- **rpt_limit_uses:** Number of uses for each of the RPTs.
- **check_ssl_certs:** Toggle on/off (bool) to check certificates in all requests. This should be forced to True in a production environment
- **use_threads:** Toggle on/off (bool) the usage of threads for the proxy. Recommended to be left as True.
- **debug_mode:** Toggle on/off (bool) a debug mode of Flask. In a production environment, this should be false.
- **resource_server_endpoint:** Complete url (with "https" and any port) of the Resource Server to protect with this PEP.
- **verify_signature:** Toggle on/off (bool) the usage of signature validation for the JWT.
- **client_id:** string indicating a client_id for an already registered and configured client. **This parameter is optional.** When not supplied, the PEP will generate a new client for itself and store it in this key inside the JSON.

- **client_secret**: string indicating the client secret for the client_id. **This parameter is optional.** When not supplied, the PEP will generate a new client for itself and store it in this key inside the JSON.

3.2.4.1.1. Default HTTP scopes

To extend the granularity of the PDP policy checks, the PEP can append extra UMA scopes to a registering resource that correspond to supported HTTP verbs. Currently, the PEP supports GET, HEAD, POST, PUT, PATCH and DELETE verbs. For the `verb_config.json` file:

- **default_scopes**: A list of default scopes that will be registered for HTTP actions, for PDP functionality. These must match the scopes existing on the Login Service's persistence repository.
- **list of scope-action associations**: for each of the scopes specified in the above list, there will be one entry with an associated action-id, e.g. `"protected_get": "get"`

3.2.4.1.2. Default resources registration

The PEP reads the definition of the default resources inserted in the database from the file located under `config/default-resources.json` of the source path, but also has its own definition under the path `charts/pep-engine/scripts/default-resources.json`.

The first option usage is mainly for a local deployment using Docker and a local image built from the um-pep-engine repository with no help of Helm Charts.

The second option is for a Helm Chart deployment which will mount the file as a volume directly into the `/data` path of the container. Notice that if this second option of deployment is followed, the unique resources for both files will be added to the database.

An example of default resources would be as follows:

```
{
  "default_resources": [
    {"name": "Sample Resource", "resource_uri": "/", "scopes": "protected_access",
    "default_owner": <uuid>},
    {"name": "Sample Resource", "resource_uri": "/workspace", "scopes":
    "protected_access", "default_owner": <uuid>}
  ]
}
```

- Mandatory Parameters:
 - **name**: String Value
 - **resource_uri**: String Value
 - **scopes**: String Value
- Optional Parameters (default values):
 - **default_owner**: String Value → `"00000000000000"`
 - **description**: String Value → `"Default description"`

3.2.4.2. Data flow

The only information the PEP handles are tickets given by the Auth Server, and RPTs which are sent only to the Auth Server.

All data is ephemeral at the time of writing, except the data stored at the config file. The resources are loaded into a mongo database, this allows to store large amounts of resources and more complex queries.

What follows is an example of the nominal flow for the PEP, and "ADES" is the Resource Server the PEP is protecting:



3.2.5. Extensibility

The design of the PEP allows for further improvements if need be. For example:

- The resource management in memory could be expanded to a full on DB, by changing how the UMA handler works internally.
- The UMA library used allows for quick implementations for resource managing, such as creation, deleting and editing.

- The proxy can be expanded to parse further parameters on top of the HTTP protocol, allowing for any kind of plugin or complex mechanism desired.

3.2.6. Applicable Resources

- UMA 2.0 Specification - <https://docs.kantarinitiative.org/uma/wg/rec-oauth-uma-grant-2.0.html>
- EOEPKA's SCIM Client - <https://github.com/EOEPKA/um-common-scim-client>
- EOEPKA's UMA Client - <https://github.com/EOEPKA/um-common-uma-client>
- EOEPKA's Well Known Handler - <https://github.com/EOEPKA/well-known-handler>
- Flask - <https://github.com/pallets/flask>

3.3. Resource Repository

3.3.1. Overview and Purpose

It is the database based on MongoDB where the resources are stored and queried for the PEP purposes

Included with the PEP there is a script at the source path that performs queries against a Mongo Database. The main purpose of this script is to reduce the usage of RAM when registering a resource locally and when querying for its content.

It is developed to generate a database called 'resource_db' in case it does not exist. The collection used for the storage of the documents is called 'resources'.

The script defines methods to:

- **Insert resource data:** Generates a document with the resource data received as input and if it already exists, it gets updated. The main parameters of the resource would be an auto-generated id provided by mongo which identify each document in the database, the resource ID provided by the login-service, and the match url which will define the endpoint of the resource. This would be mandatory parameters in order to perform other kind of queries. For updated operations, it is also capable of querying the OIDC endpoint of the Authorization Server to query if the request was performed by a valid resource operator.
- **Get the ID from a URI:** Returns the id for the best candidate of the match by a given URI.
- **Delete resources:** Receives a resource id and will find and delete the matched document, if the requesting user is a valid resource operator.

This script is manipulated by the API which would intercept the request in order to perform PUT, POST, HEAD PATCH and DELETE methods.

The GET method would be called by the reverse proxy since it will be in charge of filtering the resource with the given URI.

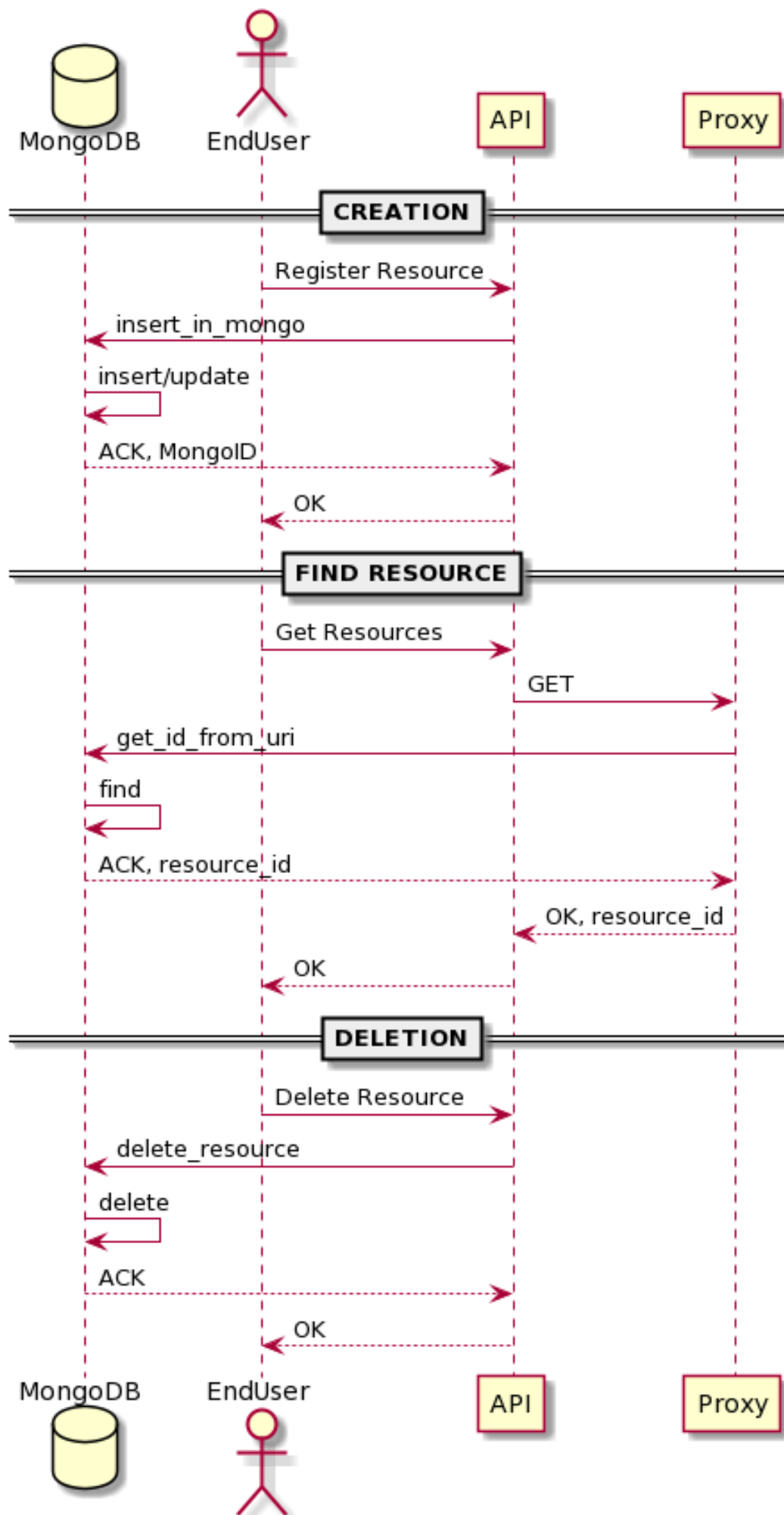
When the PEP registered a new resource this resource will contains additional scopes, one for each available HTTP action in a protected_XXX format: GET, HEAD, PUT, POST, PATCH and DELETE. These default scopes, and the associated action-ids for PDP functions, can be edited in a `verb_config.json` file.

These scopes will be used to the generation of a ticket and therefore in the PDP interface to allow or deny access to a resource.

3.3.2. Data flow

The database will only be accessible by the API or the Proxy.

The main methods for the interaction with the database are displayed in this dataflow as a summary of its scope:



3.3.3. Applicable Resources

- MongoDB image from DockerHub - https://hub.docker.com/_/mongo

3.4. Resource default Protection Policy

3.4.1. Overview and Purpose

Together with the Resource Repository, the PEP will also contact the Policy Decision Point in order to register two protection policies for the resource.

One with the scope of 'protection_read' related to the requests to the PEP endpoint with HTTP verbs HEAD and GET.

And one with the scope of 'protection_write' which is related to the requests to the PEP with the HTTP verbs PUT, POST and DELETE.

This call to `<pdp_url>/policy` will include a preset policy configuration, to be applied to the registering resource. It stands as follows:

```
{ "name": "Default Ownership Policy of <resource_id> with action <action type depends  
on type of scope> ",  
  "description": "This is the default ownership policy for created resources through  
PEP",  
  "config": { "resource_id": resource_id,  
              "rules": [ { "AND": [ { "EQUAL": { "user_name" : user_name } } ] } ]  
            },  
  "scopes": [ "protected_read" ] }
```

3.4.2. Data flow

This subroutine is triggered by the successful registration of the resource.

3.4.3. Applicable Resources

- EOEPKA's Policy Decision Point - <https://github.com/EOEPKA/um-pdp-engine>

3.5. Logging

3.5.1. Design

Logging accross the EOEPKA Building Blocks works much in the same way, by usage of a log helper class to initiate a Python logger, handler and formatter that simultaneously outputs log messages to console and a log file. These log files are set on a rotation, with a 1GB limit per each, with the 10 latest log files being kept in memory.

A new configuration yaml file is added to the building block, containing initialization parameters.

3.5.2. Log message format

INFO level log messages follow the following format:

- TIME: in ISO 8601 format, "%Y-%m-%dT%H:%M:%S%z"
- LEVELNAME: INFO by default
- COMPONENT: "PEP"
- SUBCOMPONENT: PROXY or RESOURCES
- ACTION IDENTIFIER: HTTP by default
- ACTION TYPE: HTTP method used
- LOG CODE: Unique code identifying log message type
- ACTIVITY: Detailed log message, check reference table

3.5.3. Log message codes

Subcomponent division is as follows:

- 20xx: RESOURCES
- 21xx: PROXY

Table 2. Log Codes

| Log Code | Structure |
|----------|--|
| 2001 | {"Description":"No token found/error reading token"} |
| 2002 | {"Description":"User not found in token"} |
| 2103 | {"User":user,"Description":"Token validated, forwarding to RM"} |
| 2104 | {"Ticket":ticket,"Description":"Invalid token, generating ticket for resource:"+resource_id} |
| 2105 | {"User":user,"Description":"No resource found, forwarding request for path "+path} |
| 2106 | {"User":user,"Description":"Error while redirecting to resource: "+str(exception)} |
| 2007 | {"User":user,"Description":"Returning resource list: "+resource_list} |
| 2008 | {"User":user,"Description":"No matching resources found for requested path "+path} |
| 2009 | {"User":user,"Description":"Resource created","Resource_id":resource_id,"Write Policy":write_policy_id,"Read Policy":read_policy_id} |

| Log Code | Structure |
|----------|--|
| 2010 | {"User":user,"Description":"Error occurred: "+error} |
| 2011 | {"User":user,"Description":"Operation successful","Resource":resource} |
| 2012 | {"User":user,"Description":"Resource "resource_id" deleted"} |
| 2013 | {"User":user,"Description":"No matching resources found for requested path "+path} |
| 2014 | {"User":user,"Description":"User not authorized for resource management","Resource":resource_id} |

Chapter 4. User Story Traceability

Table 3. User Stories

| User Story Code | Description | Building Block Use Case | Master Use Case |
|-----------------|---|-------------------------|-------------------------------|
| EOEPCA-12 | Reusable UMA Client Implementation | PEP-UC-001/PEP-UC-002 | EOEPCA-UC-1003 |
| EOEPCA-221 | Administrative tooling for integration and operation | - | CI/CD Task |
| EOEPCA-214 | Add Usage of Persistence Volumes | - | CI/CD Task |
| EOEPCA-35 | Ownership management for Resources | PEP-UC-005/PEP-UC-006 | EOEPCA-UC-0105 |
| EOEPCA-210 | Implementation of Helm Charts | - | CI/CD Task |
| EOEPCA-25 | Registration of Resource References | PEP-UC-005 | EOEPCA-UC-0105 |
| EOEPCA-121 | Propagation of End-User claims to the Resource Server | PEP-UC-004 | EOEPCA-UC-0106 |
| EOEPCA-205 | Create Swagger Endpoint for Resource Protection API | PEP-UC-007 | CI/CD Task |
| EOEPCA-203 | Separation of Proxy and Resource Management concerns | PEP-UC-007 | CI/CD Task |
| EOEPCA-194 | Usage of relative URLs without proxy prefix | - | CI/CD Task |
| EOEPCA-189 | Security: Verification of RPT Signatures | PEP-UC-003 | EOEPCA-UC-0107/EOEPCA-UC-0108 |
| EOEPCA-187 | Allow both RPT and ID Token Forwarding | PEP-UC-003/PEP-UC-004 | EOEPCA-UC-0106 |
| EOEPCA-178 | Default protection of resources | PEP-UC-006 | EOEPCA-UC-0104/EOEPCA-UC-0105 |
| EOEPCA-126 | Policy to Resource Data Model Extension | - | EOEPCA-UC-0108 |
| EOEPCA-173 | Implementation of strict RPT Validation Measures | PEP-UC-003 | EOEPCA-UC-0107/EOEPCA-UC-0108 |

| User Story Code | Description | Building Block Use Case | Master Use Case |
|-----------------|---|-------------------------|-----------------|
| EOEPCA-120 | Path-based resolution of Resource IDs | - | CI/CD Task |
| EOEPCA-114 | Local Registration of Resources | PEP-UC-005 | EOEPCA-UC-0105 |
| EOEPCA-99 | Command Line Interface UMA Client | - | EOEPCA-UC-1003 |
| EOEPCA-98 | Baseline Enforcement Functionality | - | CI/CD Task |
| EOEPCA-94 | Reusable UMA Client Implementation - End-User Functionality | - | EOEPCA-UC-1003 |
| EOEPCA-144 | Resource Ownership Enforcement | - | EOEPCA-UC-0105 |

<< End of Document >>